Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Miha Lunar

Porazdeljeno sledenje žarkov za upodabljanje lasersko zajetih prostorskih podatkov

diplomsko delo
na univerzitetnem študiju

Mentor: doc. dr. Matija Marolt

Ljubljana 2016

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

V diplomski nalogi preučite tehnike za realistično upodabljanje voksel svetov. Izdelajte sistem za porazdeljeno uporabljanje lasersko zajetih prostorskih podatkov. Pri tem uporabite ustrezne podatkovne strukture, ki bodo omogočale sprotno nalaganje in pomnjenje delov sveta ter učinkovito sledenje žarkov. Sistem preizkusite in ovrednotite vpliv parametrov na hitrost upodabljanja.

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Miha Lunar, z vpisno številko **63080019**, sem avtor diplomskega dela z naslovom:

Porazdeljeno sledenje žarkov za upodabljanje lasersko zajetih prostorskih podatkov

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom doc. dr. Matija Marolta,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne 5. julija 2016 Podpis avtorja:

Zahvaljujem se mentorju doc. dr. Matiji Maroltu in asistentu dr. Cirilu Bohaku za idejo, usmerjanje pri izdelavi naloge in vsesplošno mentorstvo. Velika zahvala gre tudi družini za vzpodbudo skozi vsa leta študija, zahvalil se bi pa tudi vsem kolegom in prijateljem, ki so me med študijem razvedrili in mi pomagali. Posebej bi se zahvalil še Kristini, ki mi je odprla svet in pomagala pri lektoriranju dela. Hvala pa še vsem avtorjem odprtokodnih ali drugače prosto dostopnih knjižnic, orodij, programov in sredstev, brez katerih bi bilo delo zagotovo neizvedljivo.

Kazalo

Povzetek

Diplomska naloga predstavi način upodabljanja slike s pomočjo sledenja žarkov, nekaj povezanih metod, arhitekturo interaktivnega ter porazdeljenega upodabljanja in sprotno nalaganje in pomnenje delov sveta. Upodablja se tridimenzionalne virtualne svetove igre *Minecraft* in svet, zgrajen iz oblakov točk laserskega skeniranja Slovenije. Oblaki točk so prostorsko zahtevni in v težko dostopni obliki, zato je bila razvita skripta za prenos in pretvorbo podatkov. Za hitrejšo in lažjo obdelavo podatkov je bil implementiran strežnik, ki na zahtevo v več korakih pretvori točke v obliko sveta, primerno za sledilnik. Raziskanih je bilo nekaj parametrov sledilnika in strežnika. Na koncu so prikazane še različne upodobitve, pridobljene s pomočjo omenjenih programov.

Ključne besede: sledenje žarkov, upodabljanje, voksli, LIDAR, oblaki točk, strežnik HTTP

Abstract

This thesis presents a way of rendering images through the use of ray tracing, some related methods, a highly parallel and interactive rendering architecture and on-demand world loading and caching. The ray tracer supports rendering three-dimensional voxel worlds from the Minecraft video game and a world created from a laser-scanned point cloud data set of Slovenia. Since this data demands a lot of storage space and is in an inconvenient format, a script that downloads and converts the required resources was developed. Additionally, an HTTP server was implemented for faster processing and easier data access. The ray tracer requires a specific world format, which can be generated on-demand by the server from the converted point clouds through multiple processing steps. The effects of a few configurable ray tracer and HTTP server parameters were explored. A few renders obtained through the aforementioned programs and tools are shown at the end.

Keywords: ray tracing, rendering, voxels, LIDAR, point clouds, HTTP server

Poglavje 1

Uvod

Ljudje imamo omejen pogled na svet, tako virtualen kot resničen. Naša perspektiva se ponavadi nahaja pri tleh, najbolj pogosto pa vidimo z višine le na določenih točkah, kjer so zgradbe. Sprememba pogleda ni preprosta in pogosto zahteva veliko napora ali pa uporabo zapletenih naprav, kot sta na primer letalo in helikopter. V virtualnem svetu iger je pogled ponavadi specifično načrtovan in se ga zaradi zasnove ali pa tehničnih omejitev težko spremeni. Iz želje po večji svobodi izbire pogleda se porodi potreba po posebnih programih, ki omogočajo upodobitev alternativnih pogledov na virtualen svet.

Diplomska naloga predstavi program za porazdeljeno upodabljanje svetov popularne igre *Minecraft* [36], saj je to eden izmed največjih virov ročno urejenih podatkov virtualnega sveta, ki so v velikem številu prosto dostopni. Področja se v igri ne vidi veliko, pogled pa je omejen na prvoosebno perspektivo pri tleh. Rešitev omogoča večje razdalje in višje perspektive, svet pa upodablja na bolj resničen način — s sledenjem žarkov.

Ministrstvo za okolje in prostor RS je skozi leto 2015 zagotovilo prost dostop do podatkov laserskega skeniranja celotne Slovenije [37]. Iz tega sledi ideja gradnje sveta izbranega dela Slovenije v obliki sveta igre Minecraft. Podatki Slovenije so v obliki oblakov velikega števila točk, zato so zelo prostorsko zahtevni. Pred prvo uporabo jih je potrebno najprej prenesti s stre-

žnikov ARSO[1], v prid lažji obdelavi pa je zaželena pretvorba v drugo obliko. Za to predpripravo je bil razvit "grabilnik" — orodje za prenos in pretvorbo potrebnih virov. Razvit je bil še strežnik, ki oblake točk Slovenije na zahtevo pretvarja v obliko sveta igre Minecraft, ki jo lahko sledilnik žarkov upodobi iz poljubnih pogledov.

Diplomska naloga je sestavljena iz osmih poglavij. Po uvodnem poglavju so predstavljene obstoječe rešitve upodabljanja svetov igre Minecraft oz. voksel[2] svetov in rešitve upodabljanja podatkov laserskega skeniranja. Tretje poglavje predstavi upodabljanje s sledenjem žarkov in bolj podrobno opiše strukturo virtualnega sveta Minecraft ter strukturo lasersko zajetih podatkov Slovenije. Sledijo trije glavni sklopi implementacije: v četrtem poglavju so podrobneje opisane uporabljene metode sledenja žarkov, v petem je predstavljeno delovanje grabilnika, v šestem pa gradnja sveta na strežniku. Sledi še analiza delovanja programov in pridobljene upodobitve ter sklep.

[1] Agencija Republike Slovenije za okolje.

[2] Voksel je tridimenzionalna različica piksla. Podrobneje je opisan v poglavju 3.3.2.

Poglavje 2

Pregled obstoječih rešitev

Za upodabljanje svetov igre Minecraft oz. voksel svetov obstaja več orodij, obdelava lasersko zajetih *LIDAR* podatkov pa je celo področje, saj je bolj gospodarsko koristna. Upodabljanje voksel svetov je manj poznano, zato je predstavljenih več popularnih obstoječih rešitev.

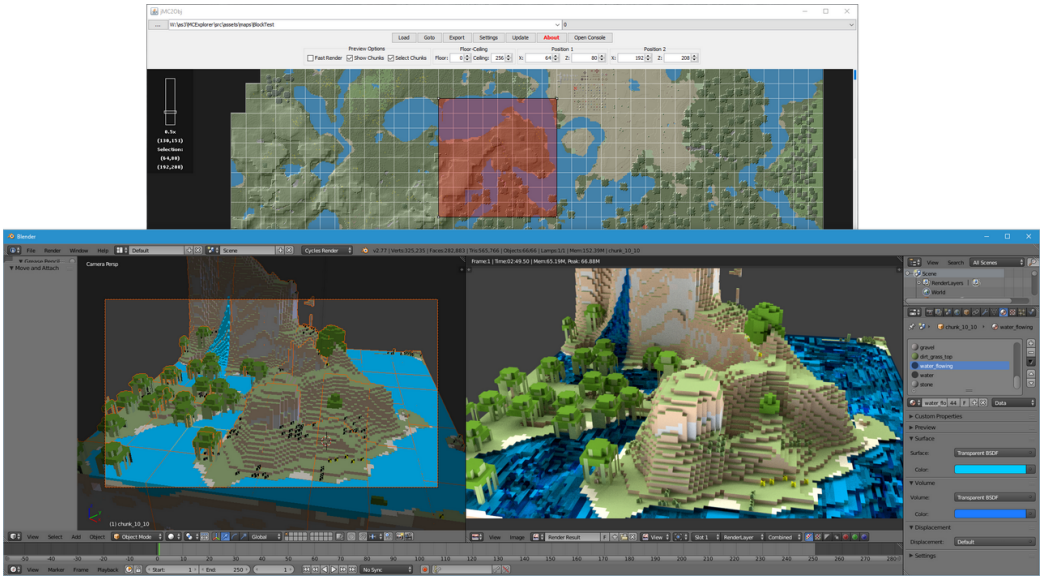
2.1 Upodabljanje voksel svetov

2.1.1 mcobj, mc2obj, jmc2obj in Blender

Orodje *mcobj* [60] je bilo med prvimi, ki so omogočali drugačen pogled v tridimenzionalni voksel svet igre *Minecraft*. Orodje samo po sebi ne upodablja ali izrisuje slik, saj služi samo pretvorbi podatkov sveta v skupek trikotnikov, predstavljenih s preprostim datotečnim formatom *OBJ*. Iz tega tudi izvira ime: *mc* — Minecraft, *obj* — format OBJ.

mc2obj [12] je bilo razvito kot nadgradnja preprostih enobarvnih blokov orodja *mcobj*. Podpira poljubne teksture blokov in tudi izračuna koordinate preslikave teksture na geometrijo. V nasprotju z orodjem *mcobj* lahko pretvori tudi bloke, ki niso v obliki kocke.

Obstaja tudi podobno orodje z imenom *jmc2obj* [10]. Ime izvira iz tega, da je orodje napisano v jeziku Java in ne Haskell kot *mc2obj*. Od teh orodij je *jmc2obj* najnovejše in ima še najbolj aktiven razvoj.



Slika 2.1: Prikaz blokov, pripravljenih z orodjem *jmc2obj* in upodobljenih v orodju *Blender*.

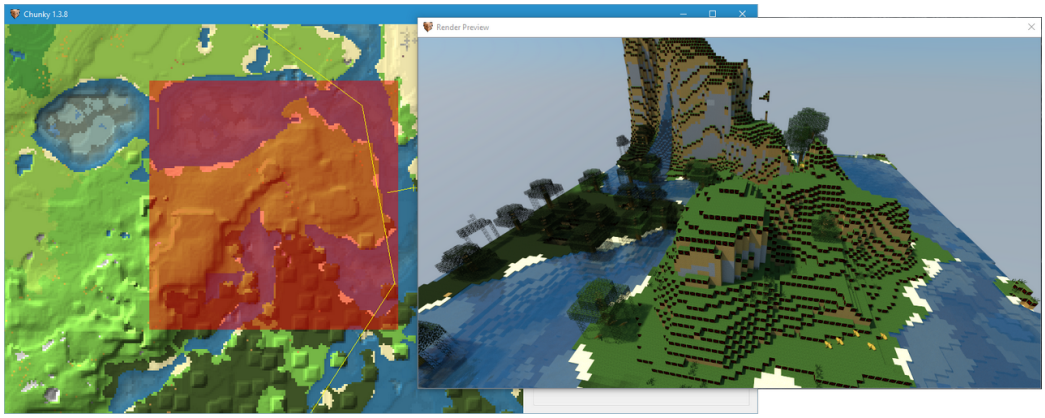
Ker ta orodja proizvedejo le geometrijo trikotnikov in koordinate morebitnih tekstur, ki se na njih nanašajo, je potrebno upodabljanje opraviti kot drugi korak s katerimkoli splošnim večnamenskim orodjem za upodabljanje, kot je na primer odprtokodno orodje *Blender* [5]. Slika 2.1 prikazuje bloke, upodobljene s pomočjo orodij jmc2obj in Blender.

S temi orodji ima uporabnik več dela kot pri celostni rešitvi, ki ima že vgrajen upodabljalnik, a so zato zelo prilagodljiva in razširljiva, saj omogočajo vse zmogljivosti, kot jih podpira orodje za upodabljanje.

2.1.2 Chunky

Ena izmed najbolj razširjenih rešitev pri upodabljanju sveta igre Minecraft je odprtokodno orodje *Chunky* [46]. Za upodabljanje strukture sveta uporablja tehniko sledenja poti, ki predstavlja bolj natančno obnašanje svetlobnih žarkov kot npr. sledenje žarkov ali klasično upodabljanje poligonov. Slika 2.2 prikazuje upodobitev v orodju.

Simuliranje poti svetlobnih žarkov, kot se pojavijo v realnem svetu, zaradi



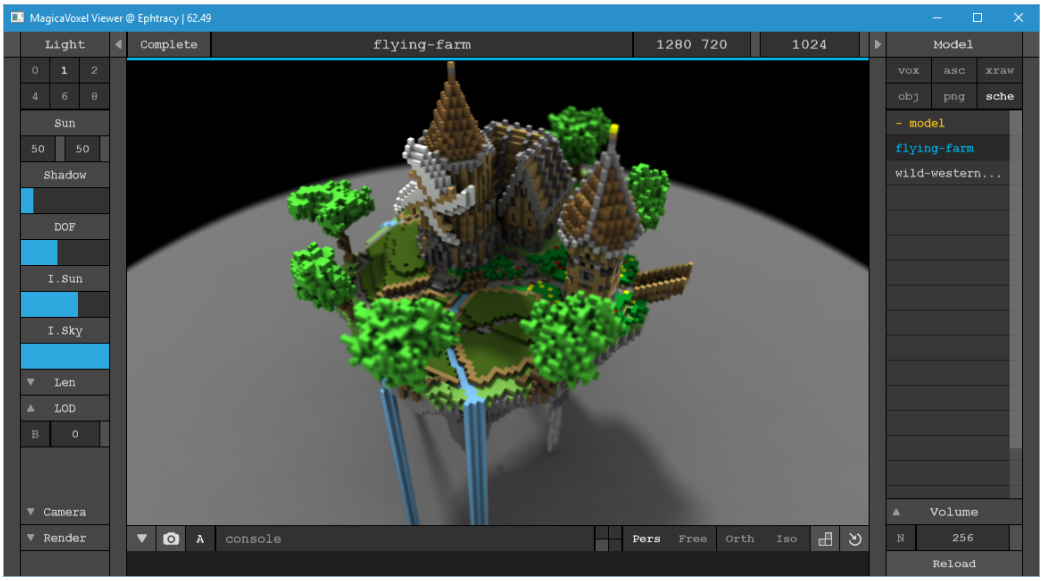
Slika 2.2: Upodobitev blokov z orodjem *Chunky*.

prevelikega števila poti ni praktično, zato se sledenje poti zanaša na statistično modeliranje z manjšim številom žarkov. Kljub temu za izris vsake slikovne pike potrebuje zelo veliko število vzorcev oz. izračunanih poti žarkov, da se končna slika dovolj približa želenemu rezultatu brez prekomernega šuma v osvetlitvi. To je ena izmed večjih pomanjkljivosti te tehnike.

Velika prednost takega upodabljanja je resničen izgled osvetlitve, saj sledenje poti upošteva večkratni odboj žarkov od vseh površin. Veliko pojavov, ki se pokažejo npr. pri motno-svetlečih prosojnih površinah, za katere je potrebno pri sledenju žarkov vložiti več truda, je zaradi narave statističnega vzorčenja lažje vključiti.

2.1.3 MagicaVoxel

Gre za novejše orodje za urejanje in upodabljanje voksel podatkov. MagicaVoxel [11] podpira uvoz in izvoz podatkov različnih vrst, med njimi tudi modele igre Minecraft. Voksle ponazori kot enobarvne kocke, upodablja pa jih s pomočjo interaktivnega sledenja žarkov. Podpira več vrst materialov in veliko nastavitev. Namenjeno je večinoma za podatke do srednje velikosti (2048^3 vokslov), upodabljanje pa je zato hitro in učinkovito. Slika 2.3 prikazuje vgrajen model, upodobljen v tem orodju.



Slika 2.3: Vgrajena "shema" `flying-farm`, upodobljena v orodju *MagicaVoxel*.

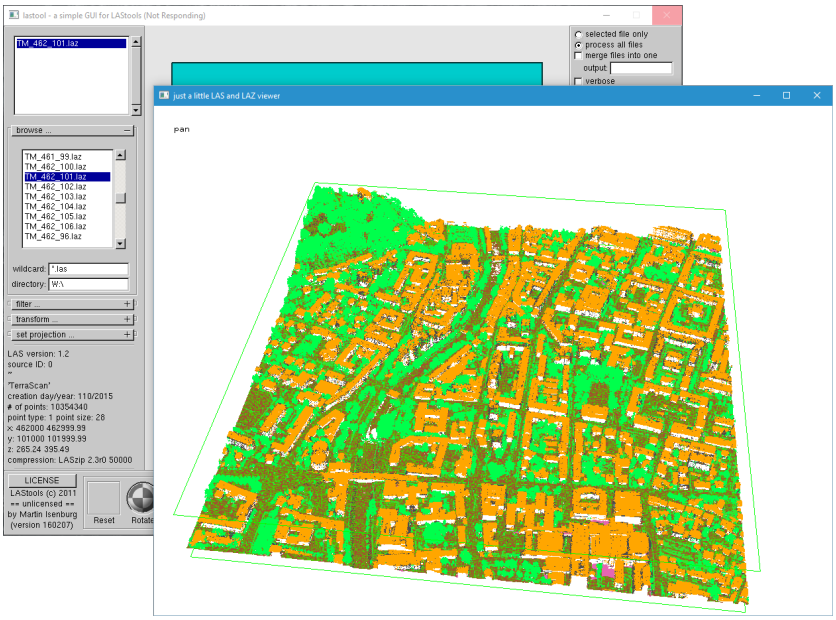
2.2 Upodabljanje LIDAR podatkov

Za upodabljanje lasersko zajetih oblakov točk oz. *LIDAR* podatkov obstaja veliko rešitev. Med njimi so direkten izris točk, izris točk s pomočjo dinamično velikih diskov, pretvorba v geometrijo trikotnikov ali v strukturo vokslov in podobno.

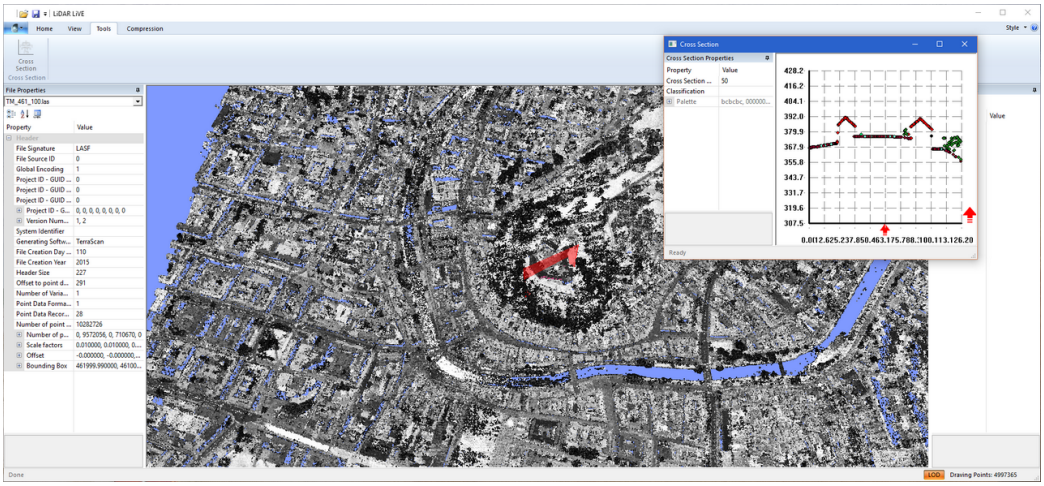
Primer zbirke orodij za obdelavo LIDAR oblakov točk *LAStools* [30] je prikazan na sliki 2.4.

Na Fakulteti za elektrotehniko, računalništvo in informatiko (FERI) je bilo razvito tudi interaktivno orodje za hiter pregled oblakov LIDAR točk z imenom *LiDAR LiVE* [32]. Točke upodablja preko direktnega izrisa s pomočjo grafične kartice, število točk pa zmanjša na obvladljivo količino s pomočjo hierarhične delitve prostora, določanja vidnega polja in stopnje podrobnosti [48]. Primer uporabe orodja je prikazan na sliki 2.5.

Diplomska naloga uporablja LIDAR podatke kot vir za generacijo voksel sveta v stilu Minecraft blokov.



Slika 2.4: Orodji *lastool* in *lasview* iz zbirke *LAStools* [30]. *lasview* prikazuje kvadrat dela Ljubljane, kjer je v ozadju grajski hrib.



Slika 2.5: Orodje *LiDAR LiVE* [32], ki prikazuje štiri kvadrate Ljubljane in presek gradu.

Poglavje 3

Teorija

Za učinkovito in resnično upodabljanje voksel sveta igre Minecraft je potrebna teoretična podlaga sledenja žarkov ter nekaj osnovnih metod. Opisana je struktura virtualnega sveta, koncept korakanja žarkov skozi voksle in prenos pridobljene barve žarka na zaslonsko sliko. Za pretvorbo lasersko zajetih oblakov točk Slovenije v voksel svet so na voljo opisani viri različnih oblik, katerih izbira je utemeljena.

3.1 Sledenje žarkov

Svetlobni žarki svojo pot začnejo pri svetlobnem viru, ki jih oddaja. Na poti se lahko večkrat razpršijo na površinah objektov in nekateri pridejo do zenice očesa, kar omogoči vid. Tako delujejo svetlobni žarki v realnosti, za simulacijo pa se pri tem potrati veliko preveč žarkov, ki gredo v smereh stran od kamere. Problem je zato pogosto obrnjen, tako da žarki začnejo pot v kameri, potujejo proti površinam in jo preko odbojev končajo v svetlobnem viru ali praznini, uporabljajo pa se tudi hibridni pristopi[1].

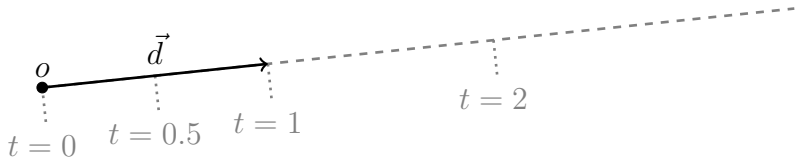
Sledenje poti se uporablja za zelo natančno modeliranje in simulacijo obnašanja svetlobe v realnem svetu, saj upošteva vso vrsto odbojev in lomov žarkov. Sledenje poti zahteva simulacijo velikega števila žarkov za sliko brez

[1]Bi-directional Path Tracing [28]

opaznega šuma in je zato zelo računsko zahtevno.

Sledenje žarkov je podobno sledenju poti, le da so pojavi v naravi pridobljeni preko bližnjic namesto simulacije zelo velikega števila žarkov. Preprostejša metoda *metanja žarkov* ne vsebuje odbojev ali loma svetlobe, zato je barva točke na sliki kar barva površine pri sečišču primarnega žarka s površino. Tako upodabljanje je povsem plosko brez kakršnihkoli svetlobnih namigov, senc, odbojev ali lomov. Sečišče se pri metodi sledenja žarkov uporabi kot izvor t. i. sekundarnih žarkov.

3.1.1 Definicija žarka



Slika 3.1: Predstavitev žarka.

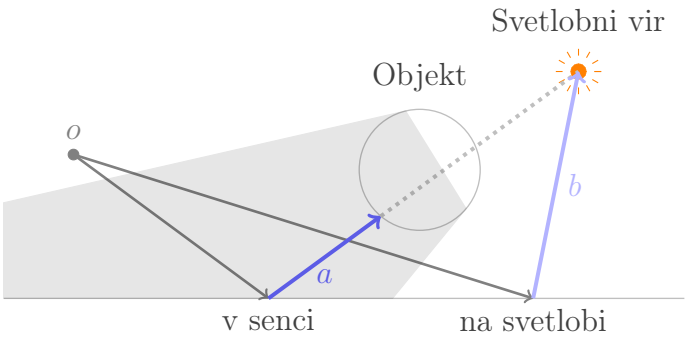
Žarek je definiran kot poltrak, ki ima izhodiščno točko o predstavljeno s položajnim vektorjem $\vec{o}$ in enotskim vektorjem smeri $\vec{d}$. Točke $\vec{r}$, ki ležijo na žarku, so s pomočjo izhodiščne točke, smeri in časa t definirane kot:

$$\vec{r} = \vec{o} + t\vec{d}$$

Čas t je klasično definiran na intervalu $[0, \infty)$, kjer je pri 0 točka enaka izhodiščni točki, pri času, večjem od 0, pa se točke nahajajo v smeri žarka. Pri implementaciji je lahko zaradi lažje uporabe čas tudi negativen, kar pomeni, da se točka nahaja na poltraku z nasprotno smerjo. Slika 3.1 prikazuje primer žarka, predstavljenega s poltrakom.

3.1.2 Senčilni žarki

Pri sledenju žarkov se za sence uporabi bližnjico *senčilnih žarkov*. Slika 3.2 prikazuje dvodimenzionalni primer z enim objektom, enim sečiščem v senci



Slika 3.2: Primer senčilnih žarkov dveh primarnih žarkov z isto izhodiščno točko o: v senci (a) in na svetlobi (b).

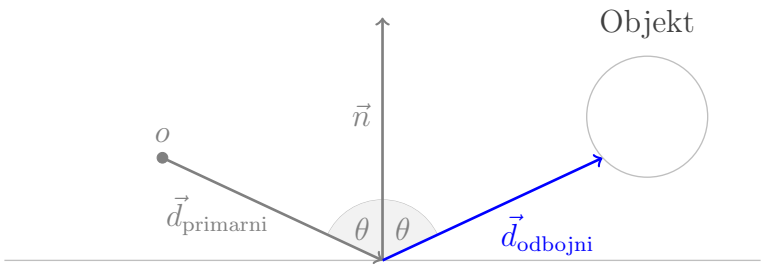
in enim na svetlobi. Pri preseku glavnega (tudi primarnega, prvega) žarka s površino se iz sečišča proti viru svetlobe pošlje senčilni žarek. Če ima senčilni žarek prosto pot do svetlobnega vira, sečišče ni v senci. V obratnem primeru, da se senčilni žarek seka s katerimkoli objektom, je sečišče v senci.

Za osnovno senčenje se nato lahko barva sečišč, ki so v senci, le izriše temneje. Sence so z uporabo enega senčilnega žarka ostre, saj se predpostavlja, da je vir svetlobe točka brez prostornine ali velikosti. Za mehkejše sence se lahko na različne načine uporabi tudi več senčnilnih žarkov.

3.1.3 Odbojni žarki

Odbojni žarki so uporabni pri upodabljanju zrcalnih ali svetlečih površin. Slika 3.3 prikazuje odboj žarka od površine v objekt. V končni rezultat točke se pri odbojnih površinah upošteva odbojno sliko v površini. Odboji se lahko ponovijo rekurzivno, tako da je vsak odbojni žarek simuliran kot nov primarni žarek naslednje stopnje rekurzije. Zaradi praktičnosti in časovne zahtevnosti je uporabno omejiti rekurzijo po številu odbojev in/ali doprinosu odboja.

Izhodiščna točka odbojnega žarka je v sečišču, smer pa je določena glede na normalo površine $\vec{n}$ in vpadno smer primarnega žarka $\vec{d}_{\text{primarni}}$, pri čemer



Slika 3.3: Primer žarka, ki se odbije s kotom θ na normalo površine $\vec{n}$ in konča na objektu.

je θ vpadni kot in $\cdot$ skalarni produkt [9]:

$$\cos\theta = -\vec{n} \cdot \vec{d}_{\text{primarni}}$$
$$\vec{d}_{\text{odbojni}} = \vec{d}_{\text{primarni}} + (2\cos\theta)\,\vec{n}$$

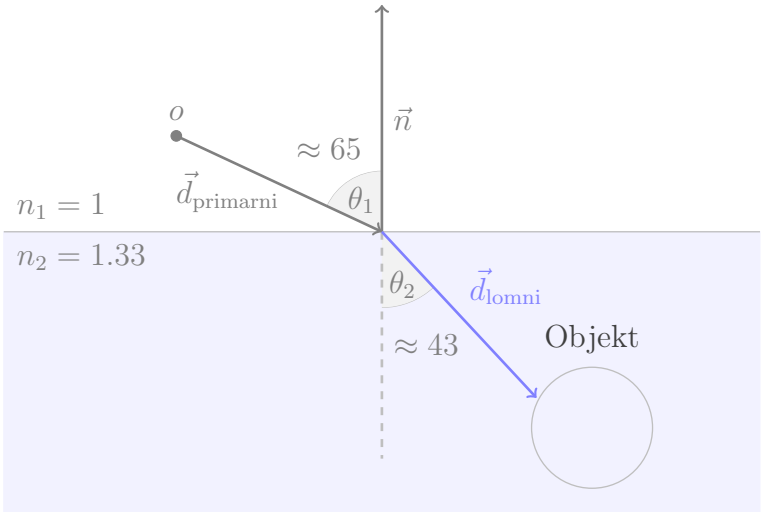
3.1.4 Lomni žarki

Prosojna snov, kot na primer voda, prepušča svetlobo skozi volumen, ki ga zavzema. Slika 3.4 prikazuje primarni žarek, ki se lomi v vodi in konča na objektu. Pri prestopu iz ene snovi v drugo nastane fenomen loma svetlobe, zaradi različne hitrosti njenega širjenja. Razmerje med hitrostjo v, lomnim količnikom n in kotom θ opisuje *Snellov zakon*:

$$\frac{\sin\theta_1}{\sin\theta_2} = \frac{v_1}{v_2} = \frac{n_2}{n_1}$$

Lomno smer pri prehodu se izračuna s pomočjo normale površine $\vec{n}$, smeri primarnega žarka $\vec{d}_{\text{primarni}}$ in lomnega količnika snovi pred prehodom n_1 in po prehodu n_2:

$$\cos\theta_1 = -\vec{n} \cdot \vec{d}_{\text{primarni}}$$
$$\cos\theta_2 = \sqrt{1 - \left(\frac{n_1}{n_2}\right)^2 (1 - (\cos\theta_1)^2)}$$



Slika 3.4: Primer lomnega žarka, ustvarjenega pri prehodu v vodo z lomnim količnikom 1.33, pod kotom 65° od normale površine $\vec{n}$, ki konča na objektu pod vodo.

$$\vec{d}_{\text{lomni}} = \frac{n_1}{n_2}\vec{d}_{\text{primarni}} - \left(\frac{n_1}{n_2}\cos\theta_1 - \cos\theta_2\right)\vec{n}$$

Pri negativnem korenu v enačbi za $\cos\theta_2$ pride do pojava *popolnega odboja*, pri katerem se vsa svetloba odbije. S tem ne pride do prehoda v drugo snov, lomni žarki pa se zato prav tako ne ustvarijo [9].

3.1.5 Fresnelove enačbe

Svetlobni žarki se ne lomijo in odbijajo enakomerno, delež enega in drugega je odvisen od vpadnega kota in lomnih količnikov snovi. Vodo se zaradi tega pojava pravokotno na površino vidi kot zelo prosojno, vzporedno pa kot ogledalo. Deleži so opisani s *Fresnelovimi enačbami* [9]:

$$R_\perp = \frac{n_2\cos\theta_2 - n_1\cos\theta_1}{n_2\cos\theta_2 + n_1\cos\theta_1}$$

$$R_\parallel = \frac{n_1\cos\theta_2 - n_2\cos\theta_1}{n_1\cos\theta_2 + n_2\cos\theta_1}$$

Pri tem sta θ_1 vpadni in θ_2 lomni kot glede na normalo površine. Lomni količnik pred prehodom je n_1, po prehodu n_2. $R_\perp$ in $R_\parallel$ sta deleža pravokotno

in vzporedno polarizirane odbite svetlobe. Pri nepolarizirani svetlobi se za delež lahko uporabi povprečno vrednost pravokotne in vzporedne polarizacije:

$$R = \frac{R_\perp + R_\parallel}{2}$$

Preostanek deleža odboja je delež prehoda oz. loma svetlobe:

$$T_\perp = 1 - R_\perp$$

$$T_\parallel = 1 - R_\parallel$$

3.2 Upodabljanje z visokim dinamičnim razponom

3.2.1 Zajem svetlobe in kontrastno razmerje

Svetilnost objektov v realnem svetu ima velik razpon, od zvezd okoli $10^{-5}\frac{cd}{m^2}$, sobne osvetlitve okoli $1-100\frac{cd}{m^2}$ do direktne sončne svetlobe okoli $10^8\frac{cd}{m^2}$ [13].

Človeško oko se prilagaja jakosti svetlobe preko številnih mehanizmov - s spreminjanjem velikosti zenice, preko kemijskih reakcij, ki spremenijo občutljivost fotoreceptorskih celic in podobno. S tem je dinamično kontrastno razmerje med najšibkejšo svetlobo, ki jo oko lahko zazna, in mejo neudobnosti okoli $10^6 : 1$ [19].

Oko pa ni zmožno zaznavanja celotnega razpona naenkrat in ima tipično kontrastno razmerje $100 : 1$ do $10\,000 : 1$ z nekaj omejitvami. Povprečni računalniški zaslon ima razmerje okoli $300 : 1$. Iz tega se je razvil koncept preslikovanja iz visokega dinamičnega razpona svetlobe realnega sveta v nizek razpon, primeren za zaslone in učinkovite datotečne formate, ki ločijo le med 256 svetlobnimi stopnjami [50].

3.2.2 OpenEXR

Za potrebe shranjevanja in obdelave slik z visokim dinamičnim razponom sta bila s strani podjetja *Industrial Light and Magic* (ILM) razvita odprt standard *OpenEXR* in programsko orodje za branje, pisanje in obdelavo slik, predstavljenih s tem formatom [45].

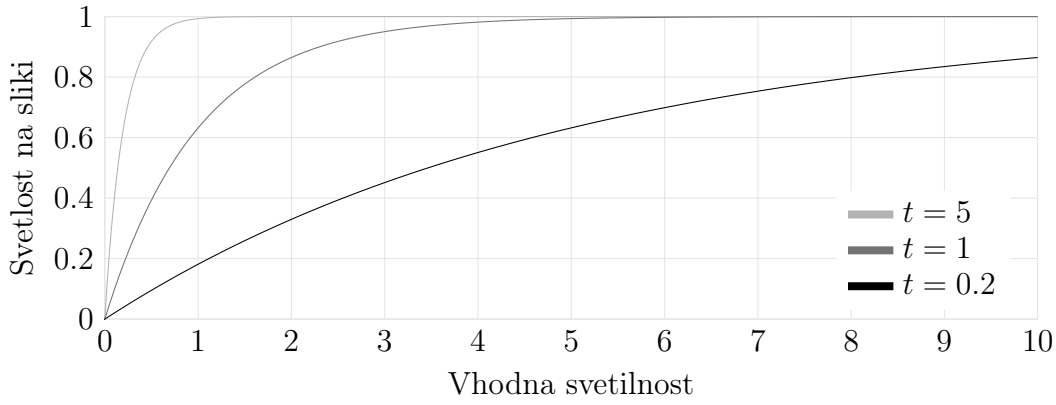
V nasprotju z datotečnimi formati, ki imajo na voljo le 256 svetlobnih stopenj, OpenEXR vsebuje podporo za 16- in 32-bitna števila s plavajočo vejico (angl. *floating point*) kot tudi poljubno število komponent oz. kanalov za vsak piksel. Zaradi razpona plavajoče vejice, fleksibilnosti formata, odprte narave in razpoložljive dokumentacije ter orodij je precej uporaben pri izdelavi programov, ki ravnajo s slikami visokega dinamičnega razpona.

3.2.3 Preslikava odtenkov

Večina prikazovalnih medijev in datotečnih formatov ima le manjši razpon, zato je pri virih z visokim razponom uporabna preslikava odtenkov (angl. *tone mapping*). Obstaja veliko različnih vrst preslikav, ki preslikajo vhodno svetilnost iz intervala $[0, \infty)$ na interval $[0, 1)$. Brez te preslikave se barve nasičijo pri vrednostih, večjih od 1, s tem pa se izgubi tudi veliko kontrasta in podrobnosti [8].

Ena izmed preprostejših preslikav izhaja iz razvoja fotografij, kjer se pri kemični sestavini odziv manjša z večjo izpostavitvijo svetlobi. Preprost model takega odziva je naravna eksponentna funkcija, prikazana na sliki 3.5. Definirana je z enačbo 3.1, kjer je x vhodna svetilnost na intervalu $[0, \infty)$, t izpostavitvena konstanta, ki določi svetlost končne slike, in $f(x)$ izhodna vrednost na intervalu $[0, 1)$.

$$f(x) = 1 - e^{-tx} \tag{3.1}$$



Slika 3.5: Primer treh odzivov eksponentne preslikovalne funkcije za svetlejšo končno sliko pri $t = 5$, vmesno sliko pri $t = 1$ in temnejšo sliko pri $t = 0.2$. Konstanta t se določi glede na razpon vhodne svetilnosti in želeno svetlost končne slike.

3.2.4 Gama popravek in sRGB

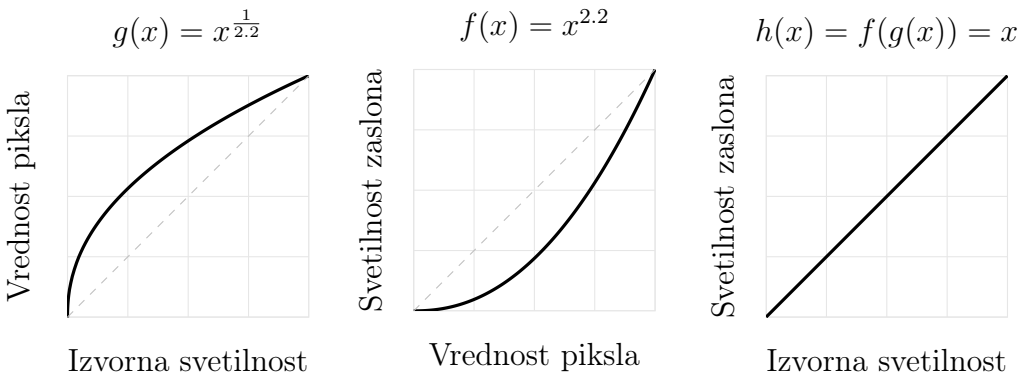
Oko je bolj občutljivo na spremembe pri temnejših odtenkih kot pri svetlejših. Nelinearna občutljivost glede na količino svetilnosti omogoča večji zaznavni razpon in prilagodljivost v sončni svetlobi. Nelinearnost odziva se lahko približno modelira s pomočjo potenčne funkcije $f(x) = x^{\gamma}$ [54].

Iz tega sledi, da je pri zajemu svetilnosti (s pomočjo kamere ali upodabljanja) vredno uporabiti več svetlobnih stopenj za temnejše dele slike, saj je tam oko bolj občutljivo. Pogosto imajo datotečni formati na voljo le majhno število diskretnih svetlobnih stopenj, zato se s pametnejšo dodelitvijo oz. s t. i. *gama stiskanjem* ali *gama kodiranjem* stopenj zviša vidna kvaliteta slike.

Vsakdanji računalniški zaslon ima prav tako nelinearno odvisno svetilnost glede na svetlobno stopnjo pikslov, ki jih prikazuje. Zasloni imajo lahko različne gama vrednosti, po večini pa se je standardizirala $\gamma = 2.2$, ki je približno obratna odzivu očesa.

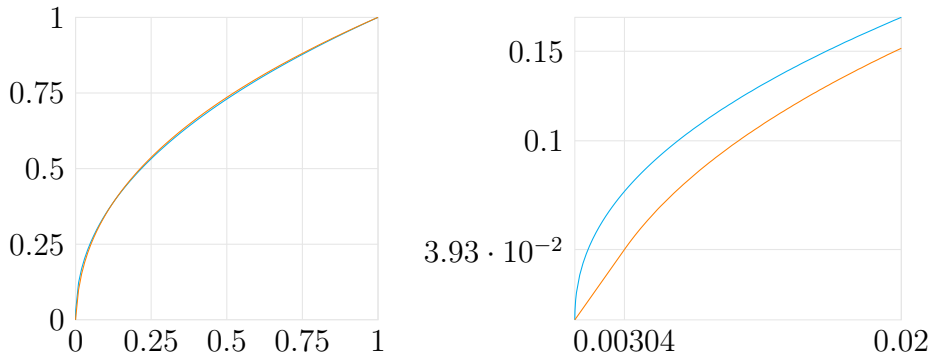
Gama popravek se izvrši pri prikazovanju gama kodirane vsebine, tako da je končna svetilnost enaka prvotni svetilnosti izvornih podatkov pred kodiranjem. Vsebina je kodirana glede na prikazovalno gamo s potenčno funkcijo $g(x) = x^{\frac{1}{\gamma}}$. V primeru, da je prikazovalna gama enaka 2.2, sledi kodirna

funkcija $g(x) = x^{\frac{1}{2.2}} = x^{0.\overline{45}}$.



(a) Gama stiskanje **(b)** Odziv zaslona **(c)** Skupni učinek

Standard sRGB, ki mu sledi večina kamer in zaslonov, definira kodiranje game nekoliko drugače kot samo s potenčno funkcijo. Kodiranje je sestavljeno iz linearnega dela pri zelo temnih vrednostih in potenčnega dela pri ostalih. sRGB sledi blizu 2.2 gama krivulji, zaradi linearnega dela pa ima potenčni del potenco 2.4 in ne 2.2 kot pri sami potenčni funkciji [53].



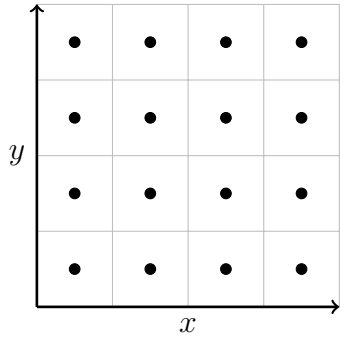
(a) Primerjava krivulj sRGB in $f(x) = x^{\frac{1}{2.2}}$.

(b) Prelom med linearnim in potenčnim delom je pri 0.00304.

3.3 Podatki v prostoru

3.3.1 Dvodimenzionalni podatki

Piksel je dvodimenzionalni ekvivalent voksla. Beseda piksel izhaja iz angleške besede *pixel* tj. *picture element* oz. slikovni element. Piksli so posamezni elementi na dvodimenzionalni enakomerni mreži, ki lahko vsebujejo več vrst podatkov ali komponent. Slika 3.8 prikazuje mrežo šestnajstih pikslov.



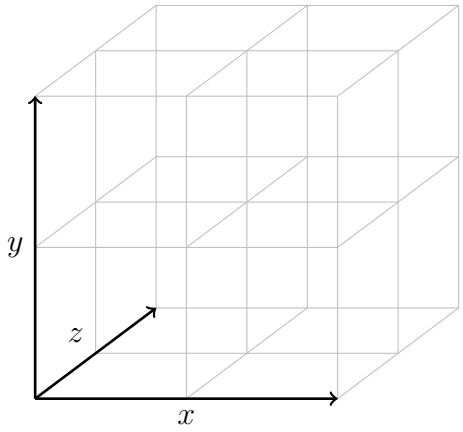
Slika 3.8: Dvodimenzionalna mreža velikosti 4×4 in 16 točk v središču pikslov.

Najbolj razširjena je uporaba t. i. RGB (angl. *Red Green Blue*) sistema, ki predstavlja točko na sliki s komponentami vsebovanosti rdeče, zelene in modre barve. Posamezni piksli so lahko predstavljeni s kvadrati, krogi, interpoliranimi barvami ipd.

3.3.2 Tridimenzionalni podatki

Voksel (angl. *voxel – volumetric element*) je točka na enakomerni mreži v tridimenzionalnem prostoru. Tako kot piksel predstavlja ploščino na mreži v dveh dimenzijah, voksel predstavlja prostornino v treh dimenzijah. Komponente voksla v prostoru lahko predstavljajo barvo, gostoto ali drugo vrednost.

V obliko, ki je primerna za prikaz, se lahko voksle prevede na več načinov. Za enostavnejšo integracijo se uporablja tehnika pridobivanja izoploskev, ki določeno vrednost vokslov predstavi kot tridimenzionalno lupino, sestavljeno iz poligonov. Bolj razširljiva metoda pa je neposredno volumetrično upoda-



Slika 3.9: Tridimenzionalna mreža velikosti $2 \times 2 \times 2$, ki prikazuje 8 vokslov.

bljanje, kjer se za sprehod skozi voksle uporabi metanje žarkov. Slika 3.9 prikazuje osem vokslov, predstavljenih s tridimenzionalno mrežo.

3.3.3 Metanje žarkov skozi voksle

Volumetrično upodabljanje polja vokslov za izris uporablja metanje žarkov skozi voksle. Za metanje žarkov se uporabi algoritem, ki obišče vse voksle, skozi katere potuje žarek. Algoritem, opisan v članku *A Fast Voxel Traversal Algorithm* [2], uporablja le pogojne stavke in prištevanje, zato je preprost pri uporabi in izračunu.

Začetno stanje

Vzpostavitev začetnega stanja se izvede v štirih delih in je odvisna od začetnega položaja in smeri žarka.

$$x = \lfloor \vec{o}_x \rfloor$$
$$y = \lfloor \vec{o}_y \rfloor$$
$$z = \lfloor \vec{o}_z \rfloor$$

x, y in z so koordinate v tridimenzionalnem kartezičnem koordinatnem

sistemu in predstavljajo položaj voksla, v katerem se nahaja začetna točka žarka o.

Pri zaokroževanju navzdol ($\lfloor\ \rfloor$) je potrebno poskrbeti, da se negativna števila zaokroži na najbližje manjše celo število (npr. $\lfloor -0.3 \rfloor = -1$, $\lfloor -2.7 \rfloor = -3$, $\lfloor 2.1 \rfloor = 2$), sicer bi se števila na obeh straneh osi na intervalu $[0, 1)$ in $(-1, 0)$ preslikala v isto koordinato 0.

$$\text{tMaxX} = \begin{cases} (x + 1 - \vec{o}_x)/\vec{d}_x & \vec{d}_x \geq 0 \\ (x - \vec{o}_x)/\vec{d}_x & \vec{d}_x < 0 \end{cases}$$

$$\text{tMaxY} = \begin{cases} (y + 1 - \vec{o}_y)/\vec{d}_y & \vec{d}_y \geq 0 \\ (y - \vec{o}_y)/\vec{d}_y & \vec{d}_y < 0 \end{cases}$$

$$\text{tMaxZ} = \begin{cases} (z + 1 - \vec{o}_z)/\vec{d}_z & \vec{d}_z \geq 0 \\ (z - \vec{o}_z)/\vec{d}_z & \vec{d}_z < 0 \end{cases}$$

Algoritem deluje na principu premikanja po koordinati, ki je najbližja robu naslednjega voksla. V spremenljivkah tMaxX, tMaxY in tMaxZ se nahaja največja časovna vrednost t, pri kateri točka po enačbi $\vec{r} = \vec{o} + t\vec{d}$ še vedno ostane v istem vokslu po osi X, Y in Z. Koordinata, katere vrednost tMax je najmanjša, je tudi najbližja.

Za koordinate voksla se zaokroži začetni položaj žarka navzdol. S tem se v vokslu $(0, 0)$ nahajajo vsi položaji s koordinatami na intervalu $[0, 1)$. Iz tega sledi, da so začetna stanja tMaxX, tMaxY in tMaxZ presek s stranico naslednjega voksla, če je smer žarka pozitivna, oz. s stranico trenutnega voksla, če je smer negativna.

$$\text{sign}(x) = \begin{cases} 1 & x > 0 \\ 0 & x = 0 \\ -1 & x < 0 \end{cases}$$

$$\text{stepX} = \text{sign}(\vec{d}_x)$$
$$\text{stepY} = \text{sign}(\vec{d}_y)$$
$$\text{stepZ} = \text{sign}(\vec{d}_z)$$

stepX, stepY in stepZ vsebujejo smer koraka po X, Y in Z koordinatah. Pri vsakem koraku se premik izvede le po eni izmed koordinat in le za en voksel, zato so step spremenljivke lahko le 1 ali -1. Pri izjemnih primerih bi lahko step vsebovala 0, a zaradi narave algoritma ostane vsaj ena pozitivna komponenta, ker je smer enotski vektor. Iz tega sledi, da se ničelne step spremenljivke ne uporabijo in lahko vsebujejo katerokoli vrednost.

$$\text{tDeltaX} = \left| \frac{1}{\vec{d}_x} \right|$$
$$\text{tDeltaY} = \left| \frac{1}{\vec{d}_y} \right|$$
$$\text{tDeltaZ} = \left| \frac{1}{\vec{d}_z} \right|$$

tDeltaX, tDeltaY in tDeltaZ predstavljajo velikost spremembe t, potrebne za premik širine, višine in globine enega voksla po koordinatah X, Y in Z.

Pri enotski velikosti voksla je premik za 1 po osi X, Y in Z premik za eno širino, višino in globino voksla. S tem so vrednosti tDeltaX, tDeltaY in tDeltaZ enake absolutnim inverznim koordinatam enotskega smernega vektorja $\vec{d}$. Na primer, če je X koordinata smeri enaka -0.25, se je potrebno

premakniti za $|\frac{1}{-0.25}| = 4$ v enotah t vzdolž smeri žarka, da se X koordinata spremeni za 1.

```
if (tMaxX < tMaxY) {
    if (tMaxX < tMaxZ) {
        side = 2;
    } else {
        side = 1;
    }
} else {
    if (tMaxY < tMaxZ) {
        side = 0;
    } else {
        side = 1;
    }
}
```

Koda 3.1: Okrnjena različica koraka, pri kateri se že vnaprej nastavi začetna stranica korakanja — spremenljivka `side`.

Uporabno je imeti shranjeno stranico voksla, preko katere se je opravil prestop – to predstavlja spremenljivka side v kodi 3.1. Korakanje je enako ne glede na predznak smeri, zato se s pomočjo te spremenljivke loči le med koordinatami X (2), Y (0) in Z (1).

Iz predznaka smeri žarka je razvidno, če je bil prestop opravljen iz pozitivne ali negativne smeri. Na primer, žarek, ki se premika desno, ne more prestopiti v voksel skozi njegovo desno stran in obratno.

Korak

Po vzpostavitvi začetnega stanja se začne korakanje, prikazano v kodi 3.2. Izvaja se, dokler se ne izpolni ustavitveni pogoj, na primer: voksel izven obravnavnega območja, prekoračeno število korakov, potrebno nadaljnje procesiranje itd.

Za posamezni korak se izvedeta dva pogojna stavka, dve prištevanji in ena prireditev, zato je relativno učinkovit.

```
if (tMaxX < tMaxY) {
    if (tMaxX < tMaxZ) {
        x += stepX;
        tMaxX += tDeltaX;
        side = 2;
    } else {
        z += stepZ;
        tMaxZ += tDeltaZ;
        side = 1;
    }
} else {
    if (tMaxY < tMaxZ) {
        y += stepY;
        tMaxY += tDeltaY;
        side = 0;
    } else {
        z += stepZ;
        tMaxZ += tDeltaZ;
        side = 1;
    }
}
```

Koda 3.2: Posamezni korak skozi voksle.

3.4 Struktura sveta igre Minecraft

Eni izmed najbolj razširjenih virov ročno urejenih voksel podatkov so svetovi, generirani ali izgrajeni s pomočjo igre *Minecraft* in povezanimi orodji. Struktura podatkov in omejitve so se skozi novejše verzije nekoliko spreminjale. Datotečni zapis sveta se je v večji meri do verzije 1.2 spremenil trikrat.

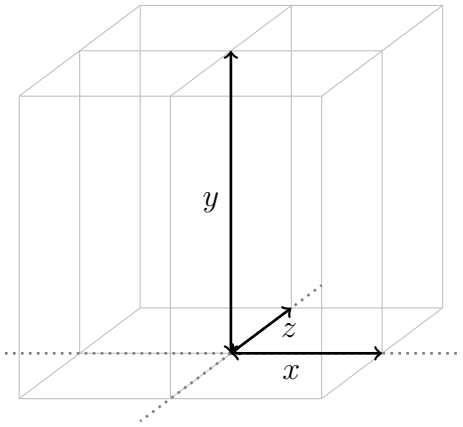
Sprva je bil razvit t. i. alfa format (angl. *Alpha level format*), imenovan po alfa razvojni stopnji igre, ki je bil kasneje nadgrajen v t. i. regijski datotečni format (angl. *Region file format*). Struktura se je še nekoliko spremenila pri naslednjem formatu *Anvil*, ki je olajšal nekatere omejitve in dodal nekaj novih funkcij.

3.4.1 Delitve blokov

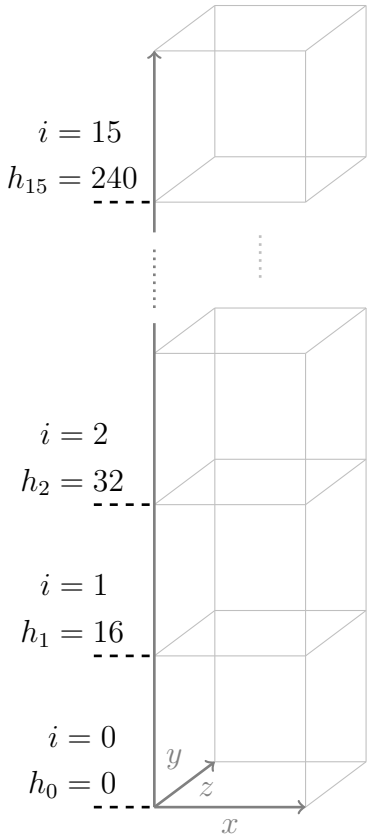
V splošnem je svet razdeljen na enakomerne kose (angl. *chunks*), ki so sestavljeni iz množice vokslov oziroma t. i. blokov (angl. *blocks*). V igri je svet avtomatsko ustvarjen tako, da ima igralec neomejeno prosto pot v katerokoli horizontalno smer, saj se, preden pride do roba trenutno ustvarjenega sveta, vnaprej sproti izdelajo novi kosi, ki se nato postavijo v svet.

S tem je svet v vse smeri neba praktično neskončen, omejen le s strani prostora pomnilniške naprave oz. trdega diska in ostalih tehničnih podrobnosti. Zaradi praktičnih razlogov, porabe pomnilnika in procesorskega časa, je po višini omejen na 256 blokov (128 pred Anvil). Vsak kos je širok in globok 16 blokov. Slika 3.10 vsebuje grobo ponazoritev štirih kosov.

Pri formatu sveta Anvil je vsak kos razdeljen še na odseke (angl. *sections*) velikosti $16 \times 16 \times 16$ blokov, kot je prikazano na sliki 3.11. Odseki se ne prekrivajo in se lahko nahajajo le na določenih višinah, zato odsek vsebuje indeks na intervalu $i = [0, 15]$, ki predstavlja lego odseka na y osi. Višina, na kateri se nahaja odsek, je tako $h = 16i$. Prazni odseki se ne shranijo.



Slika 3.10: Simbolična slika štirih kosov, vsak kos ima dimenzije $x = z = 16, y = 256$ (128).



Slika 3.11: Odseki formata Anvil z indeksi i in višinami h_i.

3.4.2 Bloki

Vsak odsek vsebuje $16 \times 16 \times 16 = 4096$ vokslov oz. blokov. Vsaka vrsta bloka ima svojo identifikacijsko številko. V igri večino vrst blokov predstavlja poenostavljeno prvino ali objekt iz realnega sveta z nekaj umetniške svobode.

Veliko blokov (npr. skalovje, zemlja, pesek) je predstavljenih stilistično v obliki kocke velikosti 1×1 metra, ki se razteza do meja vseh sosednih blokov. Nekateri bloki so predstavljeni kot ploskve s teksturo (npr. pajčevina, sladkorni trs, železnica), nekateri pa kot poligonski modeli (npr. stopnice, ograja, bakla).

Vsak blok ima v formatu sveta definirano še dodatno stanje, ki je odvisno od tipa bloka. Opisuje lahko usmerjenost, barvo, velikost ali druge lastnosti.

3.5 Struktura podatkov Slovenije

3.5.1 LIDAR

LIDAR[2] je način zajema prostorskih podatkov s pomočjo tempiranja odboja laserskih žarkov.

Ustrezno natančni senzorji izračunajo razdaljo s časovno meritvijo od osvetlitve laserja do prihoda površinskega odboja. Za večje površine je naprava, ki je sposobna takšnih meritev, nameščena na letalo. Letalo vsebuje tudi druge senzorje, ki ga pomagajo umestiti v prostor (npr. GPS, pospeškomer, žiroskop). Pridobljene razdalje so s tem georeferencirane kot točke v skupnem prostoru [21].

Kvadrati Slovenije

Ministrstvo za okolje in prostor Republike Slovenije je proti koncu leta 2015 zagotovilo javen in prost dostop do georeferenciranih in klasificiranih LIDAR podatkov za celo državo. S tem je na voljo prišel velik vir zanimivih prostorskih podatkov, primernih za upodabljanje.

[2]Light Detection And Ranging.

LIDAR podatki so na voljo na spletni strani *eVode* Agencije RS za okolje [52]. Razdeljeni so na kvadrate velikosti 1 km^2 v treh različnih zapisih [42]:

GKOT Georeferenciran oblak točk. Točke so klasificirane na tla, stavbe in na nizko, srednjo in visoko vegetacijo.

OTR Georeferenciran oblak točk, klasificiranih kot tla. Ostale točke so izbrisane.

DMR Digitalni model reliefa na osnovi točk OTR. Vsebuje pravilno mrežo višine tal ločljivosti 1 m × 1 m × 1 cm, metoda generiranja pa je bolj podrobno opisana v članku "Analiza samodejne metode za generiranje digitalnih modelov reliefa iz podatkov lidar na območju Slovenije" [38].

Vsak napis je na voljo v dveh koordinatnih sistemih [20]:

D48GK Geodetski datum, realiziran 1948, projekcija Gauß-Krüeger, referenčna ploskev Besslov elipsoid iz leta 1841.

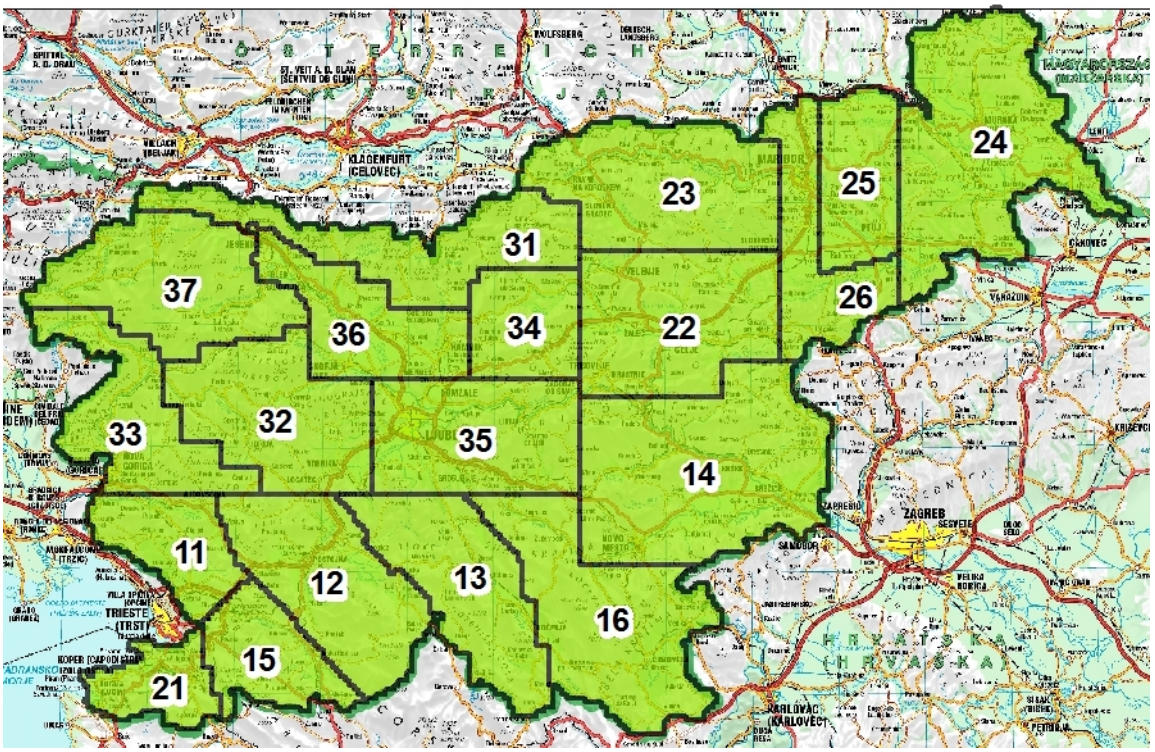
D96TM Geodetski datum, realiziran 1996, transverzalna Mercatorjeva projekcija (evropska terminologija za Gauß-Krüeger), referenčna ploskev geocentričen elipsoid GRS80, baziran na Evropskem ETRS89.

Zapis *GKOT* je zadosten, saj sta ostala zapisa le izpeljanki, ki se ju lahko izračuna sprotno. Za koordinatni sistem je bil izbran novejši D96TM. GKOT podatki so shranjeni v lastniškem zapisu zLAS podjetja Esri [34]. Kvadrati so razdeljeni na bloke območij, katerih meje so prikazane na sliki 3.12.

Posamezni kvadrati so dostopni preko parametriziranega naslova 3.1. Primer kvadrata, ki vsebuje del Ljubljanskega gradu, Tromostovje in bližnjo stran Tivolija: `http://gis.arso.gov.si/lidar/gkot/b_35/D96TM/TM_461_101.zlas`

3.5.2 Ortofoto posnetki

Agencija RS za okolje ima na voljo ArcGIS strežnik, ki ponuja veliko vrsto geodetskih podatkov. Izmed teh je na voljo tudi ortofoto zemljevid, dostopen preko parametriziranega naslova 3.2.



Slika 3.12: Bloki območij LIDAR podatkov Slovenije. Vir: ARSO [31]

`http://gis.arso.gov.si/lidar/`
`gkot/b_`*`blok`*`/D96TM/TM_`*`X`*`_`*`Y`*`.zlas`

blok Številka bloka območja, npr. `35` za Ljubljano in bližnjo okolico.

X x koordinata levega roba kvadrata v sistemu ETRS89 v kilometrih, npr. `461`.

Y y koordinata spodnjega roba kvadrata v sistemu ETRS89, npr. `101`.

gkot Tip zapisa – georeferenciran klasificiran oblak točk.

D96TM in TM Koordinatni sistem – elipsoid GRS80 s transverzalno Mercatorjevo projekcijo.

Parametriziran naslov 3.1: Deli naslova predstavljajo parametre kvadrata [25].

Posnetek se združi z LIDAR podatki tako, da se poda mejne ETRS89 koordinate LIDAR podatkov kot parametre $X_{min}, Y_{min}, X_{max}, Y_{max}$, višino in širino pa se nastavi na 1000×1000. LIDAR kvadrati so velikosti $1\,\text{km} \times 1\,\text{km}$, zato iz tega sledi, da vsak piksel posnetka pokriva navpični stolpec z zgornjo stranico $1\,\text{m} \times 1\,\text{m}$.

Primer posnetka za kvadrat `TM_461_101`: `http://gis.arso.gov.si/arcgis/rest/services/opensource_dof84/MapServer/export?bbox=461000,101000,462000,102000&size=1000,1000&bboxSR=3794&imageSR=3794&format=png&f=image`

`http://gis.arso.gov.si/arcgis/rest/services/`
`opensource_dof84/MapServer/export?`
`bbox=`X_{min}`,`Y_{min}`,`X_{max}`,`Y_{max}
`&size=`*`širina`*`,`*`višina`*
`&bboxSR=3794&imageSR=3794&format=png&f=image`

X_{min} ETRS89 x koordinata levega (zahodnega) roba želenega posnetka.

Y_{min} ETRS89 y koordinata spodnjega (južnega) roba želenega posnetka.

X_{max} ETRS89 x koordinata desnega (vzhodnega) roba želenega posnetka.

Y_{max} ETRS89 y koordinata zgornjega (severnega) roba želenega posnetka.

širina Širina želene velikosti slike v pikslih.

višina Višina želene velikosti slike v pikslih.

Parametriziran naslov 3.2: Naslov ortofoto posnetkov in opis parametrov.

Poglavje 4

Sledilnik žarkov

Razvit je bil sledilnik žarkov voksel svetov, ki voksel svet preko sledenja žarkov upodobi v zaslonsko sliko. S programom se preko uporabniškega vmesnika nadzira in upravlja celotno obdelavo in sledenje.

Sledenje žarkov je problem, ki se z lahkoto razdeli na podprobleme, saj je vsak žarek neodvisen od ostalih. Sledilnik se tako lahko porazdeli, kar pomeni, da lahko isto upodobitev obdeluje več neodvisnih programov na računalniku. Sledilnik pa podpira tudi oddaljene programe preko lokalne mreže ali interneta.

Za vir sveta se lahko uporabi ali namenske zapise igre Minecraft ali pa strežnik HTTP, ki na zahtevo iz oblakov točk zgradi želene kose Slovenije. Za potrebe hitrega predogleda se upodabljanje izvaja po delih in v več ločljivostnih stopnjah, v prid bolj resničnemu prikazu pa se uporabijo opisane metode senčenja in dodatnih efektov.

4.1 Orodja

Sledilnik žarkov in glavni uporabniški vmesnik je bil napisan v jeziku ActionScript 3 za Flash in AIR platformi, kar omogoča tudi uporabo prek spleta. Za razvoj je bilo uporabljeno odprtokodno orodje FlashDevelop [15], ki kodo prevaja s pomočjo odprtokodnega razvojnega paketa FlexSDK [1]. Za izde-

Slika 4.1: Uporabniški vmesnik sledilnika žarkov z upodobljenim prizorom nekaj blokov.

lavo poligonskih tridimenzionalnih modelov je bilo uporabljeno odprtokodno orodje Blender [5].

4.2 Uporabniški vmesnik

Za potrebe sledilnika je bil razvit sistem uporabniškega vmesnika, ki predstavi različne nastavitve v vrstici nad upodobljeno sliko. Vmesnik omogoča nadzor poteka upodabljanja in spreminjanje nastavitev različnih funkcij upodabljalnika. Po končani upodobitvi se lahko preko vmesnika shrani še končna slika v formatu *PNG* ali *OpenEXR*. Na sliki 4.1 je prikazan uporabniški vmesnik z upodobljenim prizorom blokov.

4.3 Sprotno nalaganje in pomnenje sveta

Svetovi igre Minecraft in pretvorjeni lasersko skenirani podatki Slovenije so veliki in zelo prostorsko zahtevni, zato se jih lahko v pomnilnik naloži le po

delih oz. t. i. *kosih*. Za hitrejši dostop do podatkov in optimizirano prostorsko zahtevnost se zgradi in upravlja več stopenj hierarhije predpomnilnikov.

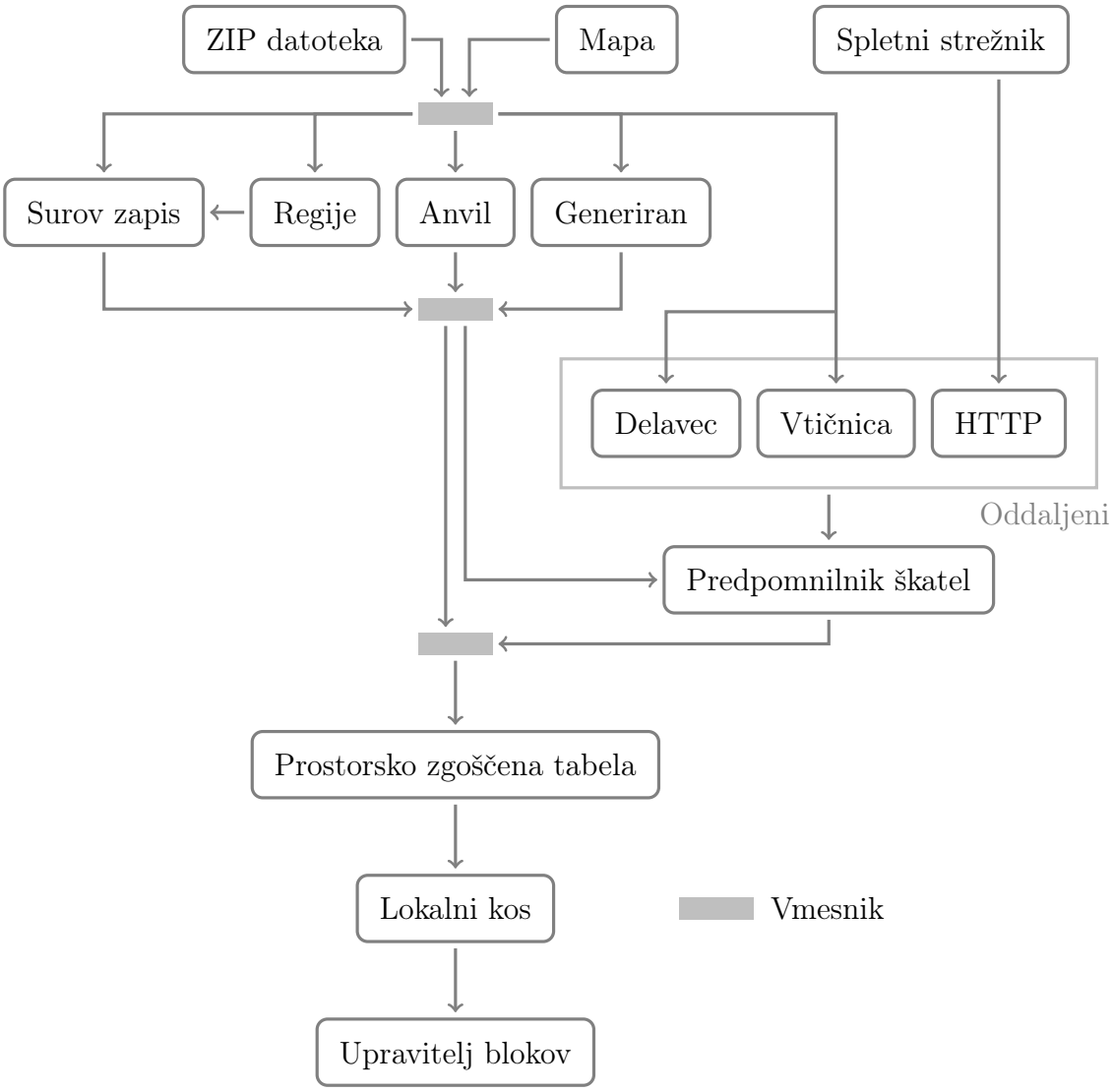
Sledilnik vsebuje možnost nalaganja datotek sveta iz prebrane stisnjene ZIP datoteke, direktno iz datotečnega sistema, tj. iz mape na disku ali pa preko prenosa s spletnega strežnika. Dostop preko teh dveh načinov je abstrahiran, tako da način dostopa ne vpliva na ostale sisteme. Slika 4.2 prikazuje abstrakcije in potek nalaganja kosov.

Obstaja več možnih zapisov sveta. Surov zapis kosov shrani vsak kos kot svojo datoteko. Format regij, zaradi velikega števila datotek pri surovem zapisu, združi več kosov skupaj v eno datoteko oz. regijo. Format Anvil je zadostno spremenjen, da stoji sam zase, še vedno pa uporablja zelo podoben surov zapis kosov, tako kot regije, v katerih se nahaja.

Sledilnik dodatno podpira tudi interno generiran svet brez datotečnega zapisa. Svet vsebuje platformo, na kateri so prikazani vsi tipi blokov, okoli nje pa so generirani hribi, pri katerih je za vir višine blokov uporabljen šum tipa Simplex [17]. V celoti je generiran sprotno in na zahtevo, za pospešitev pa uporablja prostorsko zgoščeno tabelo (angl. *spatial hash table*) za vsako koordinato bloka. Zaradi omenjenih lastnosti je uporaben pri testiranju novih blokov in upodabljanju v razmerah, kjer zunanji viri sveta niso na voljo.

Razvit je bil še lasten pomnilniški format *škatel*, ki nima datotečne predstavitve, združi pa lahko več kosov v eno t. i. škatlo za hitrejšo obdelavo in lažji prenos v oddaljen svet. Za uporabo škatel se vsak zapis najprej prevede v pravo obliko, kar vzame nekaj več začetnega procesiranja. V zameno pa je sistem zato bolj nadzorljiv in predvidljiv. Sistem oddaljenih svetov ima poseben potek izvajanja, uporablja pa obliko škatel za prenos podatkov o svetu.

Pri kosih in škatlah se za potrebe hitrejšega dostopa uporabi prostorsko zgoščena tabela, z razpršilno funkcijo predstavljeno v kodi 4.1. Zadnji kos in njegove koordinate se obdrži za hiter lokalen dostop brez dodatnega procesiranja pri zgoščeni tabeli.



Slika 4.2: Diagram abstrakcij in poteka nalaganja kosov sveta.

4.3.1 Pomnenje kosov

Za vse kose ali škatle se za potrebe uravnavanja pomnilnika uporablja urejen povezan seznam. Urejen je tako, da so proti koncu seznama nedavno obiskani kosi, proti začetku pa starejši kosi.

Pomnilnik kosov je omejen, zato se pri prostorski stiski na začetku odreže ustrezno število kosov, da se sprosti dovolj prostora. S tem se v predpomnilniku ohrani več nedavno uporabljenih kosov, kar je v nekaterih primerih

```
zx = bx & 0xFFFF
zy = bz & 0xFFFF
zx = (zx | (zx << 8)) & 0x00FF00FF
zx = (zx | (zx << 4)) & 0x0F0F0F0F
zx = (zx | (zx << 2)) & 0x33333333
zx = (zx | (zx << 1)) & 0x55555555
zy = (zy | (zy << 8)) & 0x00FF00FF
zy = (zy | (zy << 4)) & 0x0F0F0F0F
zy = (zy | (zy << 2)) & 0x33333333
zy = (zy | (zy << 1)) & 0x55555555
h = zx | (zy << 1)
```

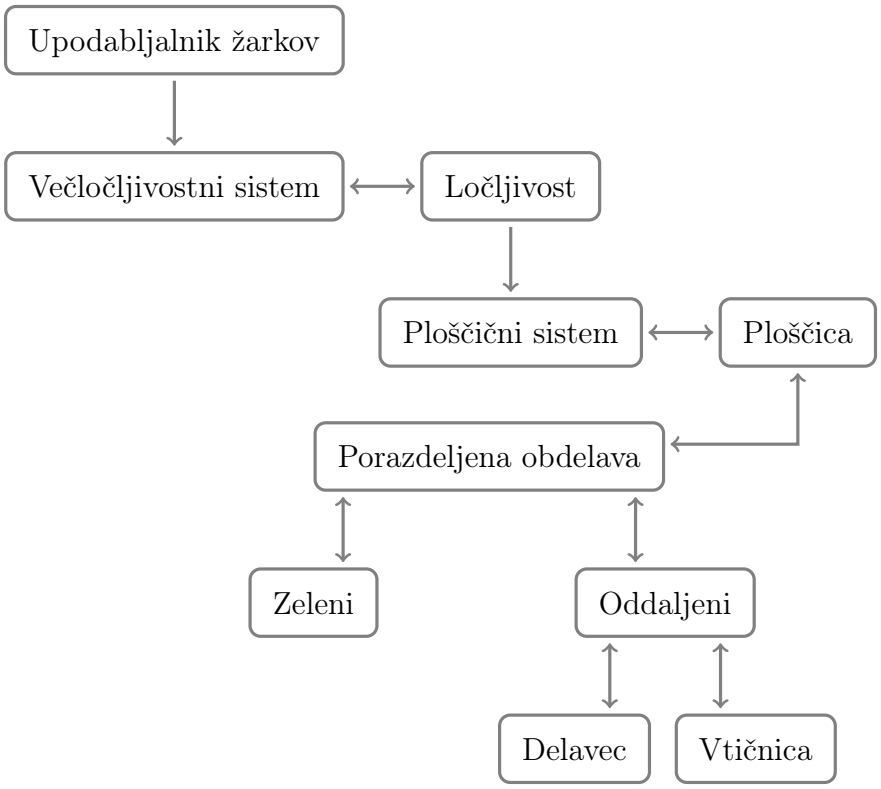
Koda 4.1: `&` predstavlja bitno operacijo "in", `|` bitni "ali", `«` pa bitni premik v levo. `bx` in `bz` sta 32-bitni celi števili, ki predstavljata koordinate kosa, `h` pa izhodna razpršilna vrednost – indeks tabele. Ta način "prepletanja" bitov koordinat se imenuje Mortonova koda ali krivulja Z reda, ima pa to dobro lastnost, da v prostoru bližnje koordinate preslika v bližnje enodimenzionalne vrednosti [3].

lahko bolj učinkovito kot brisanje po istem vrstnem redu, kot so bili kosi dodani.

4.4 Upodabljalnik žarkov

Upodabljanje se izvaja preko več nivojev. Sistem ima možnost upodabljanja skozi ločljivostne stopnje, od hitrejšega predogleda manjše ločljivosti do ostre slike končne ločljivosti. Vsaka ločljivostna stopnja se razdeli še na mrežo ploščic, ki jih lahko upodobi več različnih vrst obdelovalcev. Na sliki 4.3 so prikazani nivoji upodabljanja. Upodabljalnik prikaže vsak blok glede na tip kot kocko, model tridimenzionalnih poligonov ali pa kot ravnine v prostoru, na katere se preslika izbrana tekstura. Za primere upodobitev se uporabi paket tekstur z imenom *piehole*[1].

[1] *Piehole* je pod licenco CC BY 3.0 avtorja Alexa Völka. [58]

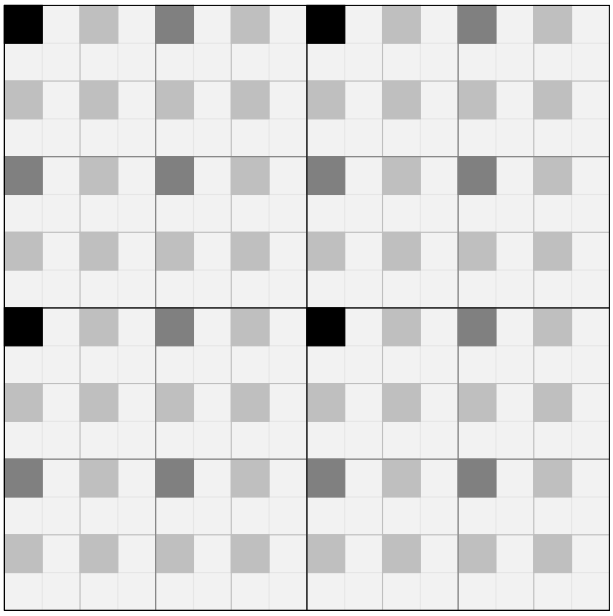


Slika 4.3: Stopnje in abstrakcije sistema za upodabljanje žarkov.

4.5 Večločljivostno upodabljanje

Za potrebe hitrega predogleda se slika upodobi postopoma, najprej z majhno ločljivostjo, nato pa z vedno večjimi, do končne ločljivosti. Pomanjšane ločljivosti so izbrane na tak način, da se vsako gručo štirih pikslov nadomesti samo z zgornjim levim pikslom. Z vsako večjo ločljivostjo se dodajo ostali trije piksli, predstavljeni eno stopnjo svetleje na sliki 4.4. S tem se upodobi enako število pikslov, kot se bi samo za končno ločljivost. Izguba nastane le zaradi manj optimalne rabe predpomnilnikov in minimalnih dodatnih stroškov sistema.

Pri sprotnem predogledu so manjše ločljivostne slike povečane na velikost končne slike, manjkajoči piksli pa so interpolirani z vgrajenim filtrom grafičnega pogona. Manjša začetna ločljivost omogoči hitrejšo navigacijo in manipulacijo, saj je hiter predogled, kljub slabši kvaliteti, pogosto dovolj za



Slika 4.4: Štiri ločljivostne stopnje, prikazane z odtenki na končni sliki dimenzij 16×16. Višje resolucije so predstavljene s svetlejšimi piksli.

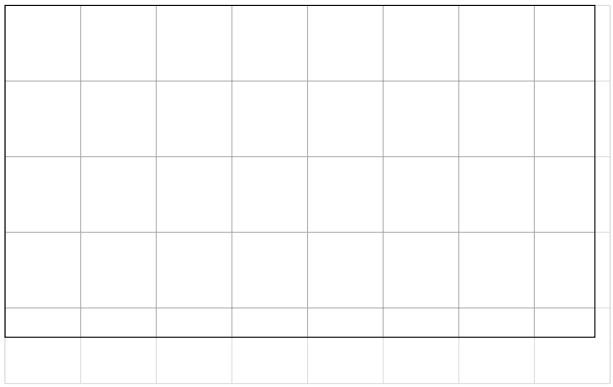
naslednjo odločitev pri spreminjanju nastavitev.

4.6 Ploščični sistem

Vsaka ločljivostna slika se razdeli na ploščice (angl. *tiles*, tudi vedra, angl. *buckets*). Ploščica je kvadrat pikslov določene velikosti. Pri dimenzijah slike, ki niso večkratne velikosti ploščice, je potrebno obravnavati delne ploščice tako, da se upoštevajo le vidni piksli, kot je prikazano na sliki 4.5. Skupina ploščic se upodablja zaporedoma po določenem vrstnem redu.

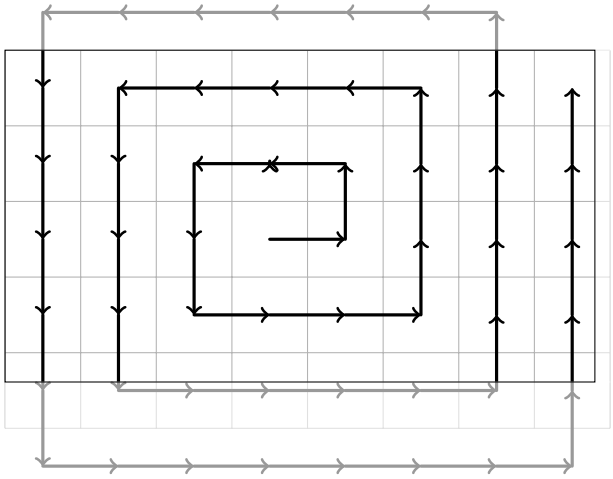
Eden izmed vrstnih redov, primernih za hiter predogled slike med upodabljanjem, je spiralni vrstni red, prikazan na sliki 4.6. Začne se s ploščico v središču slike in se nadaljuje v obliki oglate spirale proti robovom slike. Ploščice, ki ležijo na spirali, a niso del slike, se lahko prezre. Osrednji del slike se pri tem vrstnem redu upodobi najprej, saj je za postavitev kamere in predogled nastavitev bolj pomemben kot robovi.

Spiralni vrstni red je nekoliko počasnejši zaradi predpomnilnikov, zato je

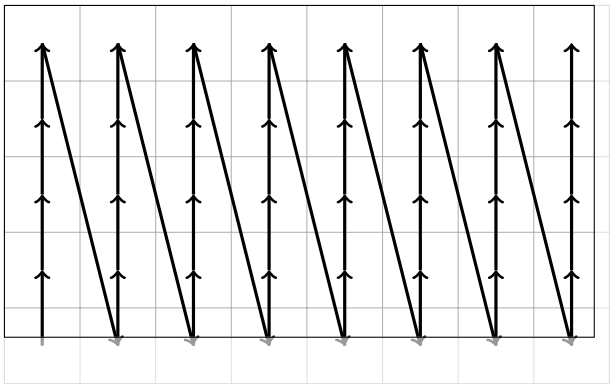


Slika 4.5: Prikaz razdelitve slike razmerja 16:9 na enakomerno velike ploščice.

uporaben tudi navpičen vrstni red, prikazan na sliki 4.7. Ta vrstni red je hitrejši tudi zaradi same strukture sveta, ki je sestavljen iz navpičnih kosov. S tem je pri navpičnem vrstnem redu manj preklapljanja med kosi kot pri spiralnem, kjer se ploščice upodabljajo tudi na horizontalni poti.



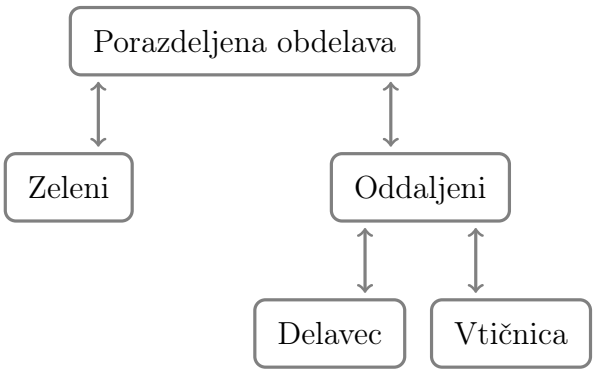
Slika 4.6: Spiralni vrstni red upodabljanja ploščic.



Slika 4.7: Navpični vrstni red upodabljanja ploščic.

4.7 Porazdeljena obdelava

Ploščice lahko upodobijo različni tipi obdelovalcev. Ploščice se ustvarjajo na zahtevo glede na proste obdelovalce. Dodeljevanje ploščic obdelovalcem ureja sistem za porazdeljeno obdelavo (angl. *distributed processing system*). Sistem je neodvisen od tipa obdelovalcev in je odporen na njihov izpad, saj med obdelavo zahteva sprotna poročila o stanju.



Slika 4.8: Diagram abstrakcij porazdeljene obdelave.

4.7.1 Zeleni obdelovalci

Najpreprostejši tip obdelovalca je t. i. *zeleni obdelovalec* (angl. *green processor*). Imenovan je po t. i. *zeleni niti* (angl. *green thread*), ki v programu simulira vzporedno obdelavo podatkov. To niso prave niti, kot jih ponuja operacijski sistem, saj jih ureja program sam. Zeleni obdelovalci se tako obdelujejo lahko samo v eni pravi niti in posledično samo na enem jedru fizičnega procesorja.

Zato da se pri zeleni obdelavi ostali deli programa med upodabljanjem ne ustavijo, mora imeti zeleni obdelovalec določeno časovno rezino, v kateri se lahko izvaja. Za potrebo osvežitve uporabniškega vmesnika 60 sličic na sekundo sledi, da mora svojo obdelavo prekiniti po največ $1\,\mathrm{s}/60 = 16.\overline{6}\,\mathrm{ms}$. Tudi ostali deli porabijo nekaj časa, zato je 15 ms primerna rezina časa, ki ne kvari interaktivnosti programa.

Pri tako majhni rezini so lahko izgube pri prekinjanju upodabljanja in preklapljanju med različnimi deli programa velike, zato se z večjo rezino časa lahko pohitri obdelavo, pri čemer trpi odzivnost uporabniškega vmesnika.

4.7.2 Oddaljeni obdelovalci

Zeleni obdelovalci motijo ostale dele programa, saj se izvajajo na isti niti, iz česar prihaja potreba po ločenem delu za obdelavo. Oddaljeni obdelovalci uporabljajo sistem sporočil, preko katerih se uravnava potek upodabljanja. Prenašajo različne informacije, tako status kot tudi stanje, potrebno za upodobitev ploščice.

Druga stran oddaljenega obdelovalca se lahko nahaja na drugi niti. Lahko pa je tudi ločen program na istem ali pa preko internetne povezave oddaljenem računalniku. Iz tega sledita tudi dva sistema sporočil, delavski (angl. *worker*) in vtičnični (angl. *socket*).

Delavski sistem

Izvajalno okolje Flash Player od verzije 11.4 dalje podpira izvajanje na več nitih (in s tem procesorskih jedrih) preko t. i. delavcev (angl. *workers*). Za vsakega delavca se ustvari svoje izvajalno okolje, zato so bolj sredstveno zahtevni kot tradicionalne niti, omogočajo pa upodabljanje na več jedrih v enem programu [14].

Sporočilni sistem med glavnim programom in delavci je realiziran s pomočjo vgrajenih sporočilnih kanalov (angl. *message channels*), ki urejajo sinhronizacijo in prenašanje podatkov med nitmi.

Vtičnični sistem

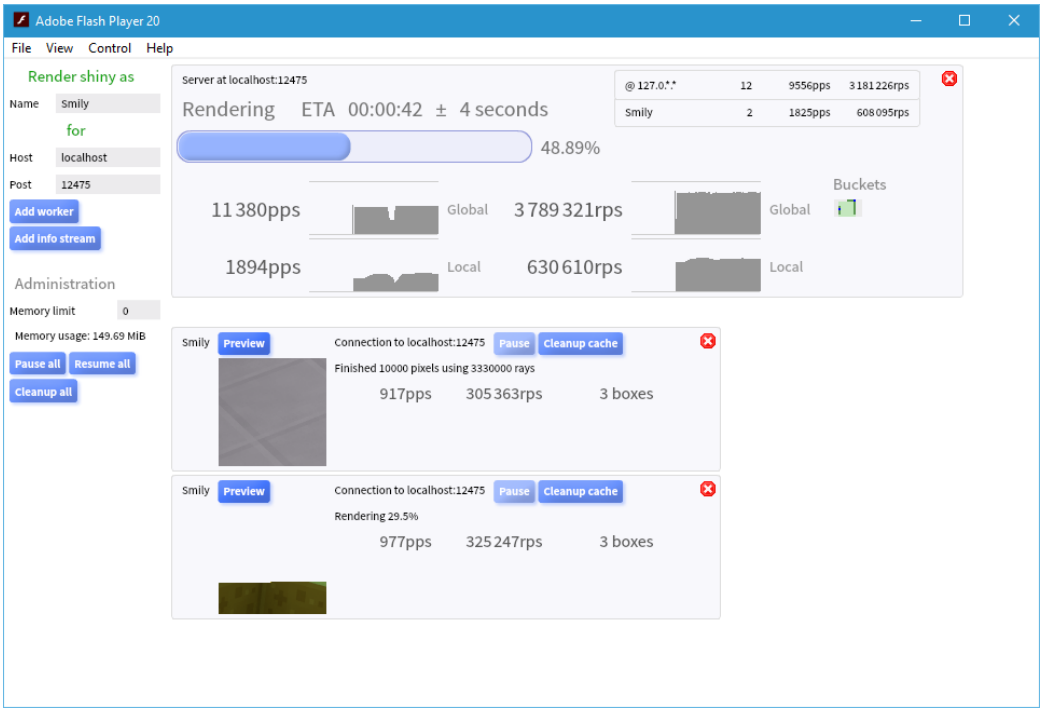
Glavni strežniški program in obdelovalci so ločeni do te mere, da obdelovalec vse potrebne podatke dobi preko sporočil. Iz tega sledi, da se upodabljanje lahko izvaja tudi preko internetne povezave, kjer se uporabi sistemske TCP vtičnice (angl. *TCP sockets*) za mrežne povezave.

Podprogram sledenja žarkov

Za potrebe oddaljenega upodabljanja je bil ustvarjen podprogram sledenja žarkov, ki uporablja le minimalne razrede, potrebne za upodabljanje. Podprogram se lahko kot delavec naloži v izvajalno okolje glavnega programa, kjer vzpostavi povezavo preko delavskega sistema in vgrajenih sporočilnih kanalov.

Pri samostojnem zagonu se preko vtičnice poveže na privzet lokalni naslov in vrata, saj sam po sebi nima uporabniškega vmesnika. Za potrebe lažje in širše uporabe je bil razvit tudi prijaznešji uporabniški vmesnik, preko katerega se lahko kdorkoli poveže na glavni program.

Uporabniški vmesnik, prikazan na sliki 4.9, preko podprograma ponuja možnost povezave na poljubne naslove, upodabljanja dobljenih ploščic in prikazovanja statusa glavnega programa.



Slika 4.9: Uporabniški vmesnik podprograma sledenja žarkov med upodabljanjem.

4.8 Preskakovanje kosov

Kosi se raztezajo le po horizontalni ravnini sveta. Iz tega sledi, da so vsebovani voksel podatki v veliki meri ravni. Zgornji del kosa je pogosto prazen oz. ima tip zraka. Oddaljen pogled na svet z višine s tem vsebuje veliko korakanja skozi prazne voksle.

Kamera se lahko nahaja izven polja vokslov, zato se žarke na začetku premakne vzdolž smernega vektorja do zgornjega roba kosov sveta. Žarki lahko tudi povsem preskočijo sledenje, če se nahajajo pod ravnino sveta in so usmerjeni navzdol oz. če so nad kosi in so usmerjeni navzgor.

Za preskok praznih delov kosov je potrebna informacija o najvišjem bloku v kosu, ki se za vsak kos izračuna v času nalaganja. Ob znanem najvišjem bloku se praznino predstavi s kvadrom, ki se spodaj dotika bloka, ob straneh pa roba kosa. Vrh kvadra je določen z vrhom sveta oz. se niti ne rabi posebej upoštevati, če se stranice raztezajo navpično proti neskončnosti.

4.9 Okoliško senčenje

Pri sledenju žarkov ni naravnega senčenja kot pri sledenju poti, zato se za bolj prepričljive slike uporabi veliko približkov. Eden izmed teh približkov je t. i. okoliško senčenje (angl. *ambient occlusion*), ki senči površino glede na geometrijo okolice. Sliki 7.14 in 7.15 prikazujeta primerjavo med različnimi načini in parametri okoliškega senčenja.

4.9.1 Grobo voksel senčenje

Pri geometriji z voksli obstaja način zelo prepostega in grobega, a hitrega okoliškega senčenja. Okoli praznega voksla tik pred presekom žarka se poizvede o sosednih vokslih. Zaradi preprostosti se upošteva le osem sosednih vokslov okoli osi normale sečišča. Glede na to, če so sosednji voksli prazni (tj. imajo tip zraka) ali ne, se izračuna preprost sivinski gradient.

4.9.2 Senčenje s pomočjo okoliških žarkov

Senčenje, ki se ozira samo na nekaj sosednjih vokslov, ima veliko pomanjkljivosti in je dovolj le za preproste potrebe hitrega prepoznavanja globine. Za bolj prepričljiv izgled okoliškega senčenja se uporabi senčenje z okoliškimi žarki.

Iz sečišča se v hemisferi okoli normale na površino v naključne smeri pošljejo okoliški žarki, ki imajo lahko za potrebe hitrosti tudi omejen potovalni čas oz. dolžino. Okoliški žarki so sorodni senčilnim žarkom, saj se zanje upošteva le, če sekajo geometrijo ali ne. Iz vseh okoliških žarkov se za točko izračuna vrednost zakrivanja.

Če se žarek na poti ne seka z geometrijo oz. če se bi sekal po določeni dolžinski omejitvi, potem vrednost zakrivanja ostane nespremenjena, saj se upošteva, kot da je v smeri žarka povsem odprt prostor. V primeru sekanja žarka pa se vrednost zakrivanja poveča za $1 - \frac{t}{\text{razpon}}$, kjer je t čas sečišča oz. dolžina žarka od izhodiščne točke do sečišča in razpon dolžinska omejitev, s katero je senčenje obteženo.

Sledi, da bližnji predmeti prispevajo več k vrednosti zakrivanja kot oddaljeni predmeti. Interval zaloge vrednosti zakrivanja je tako $[0, n]$, kjer je n število okoliških žarkov. Svetilnost točke se nato spremeni na naslednji način:

$$m = 1 - k_m \frac{z}{n}$$

$$s_n = \begin{cases} sm & m > 0 \\ 0 & m \leq 0 \end{cases}$$

Nova svetilnost s_n, ki upošteva okoliško senčenje, se izračuna s pomočjo originalne svetilnosti s, konstante moči senčenja k_m, vrednosti zakrivanja z in števila okoliških žarkov n. Množitelj svetilnosti m je lahko zaradi konstante moči negativen, zato je potrebno svetilnost omejiti s spodnjo mejo 0. S pomočjo konstante moči se lahko poveča ($k_m > 1$) ali pomanjša ($k_m < 1$) učinek senčenja po potrebi.

4.10 Razsvetljava z bloki

Svet poleg sonca vsebuje tudi posamezne vire svetlobe, predstavljene s posebnimi bloki. Vir razsvetljave je lahko poseben zapis v formatu sveta ali pa pridobitev svetilnosti s pomočjo senčilnih žarkov. Na sliki 7.16 je prikazana primerjava med tema načinoma razsvetljave.

4.10.1 Razsvetljava s senčilnimi žarki

Za natančno senčenje in prepričljiv izgled se uporabi senčilne žarke. Okoli voksla, kjer se nahaja točka sečišča, se v vseh treh dimenzijah pregleda bloke v določenem razponu. Sproti se za vsak vnaprej določen tip svetlobnega bloka shrani njegovo mesto v prostoru.

Po odkritju vseh bližnjih svetlobnih blokov se iz sečišča proti vsakemu pošlje senčilni žarek. V primeru da se žarek ustavi, preden prispe do svetlobnega bloka, se svetilnost točke sečišča ne spremeni, ker je sečišče v senci. Če ima žarek prosto pot od sečišča do bloka, se svetilnosti doda vrednost osvetljevanja. Ta je odvisna od skalarnega produkta normale površine sečišča in smeri žarka ter od obratno-kvadratne oddaljenosti vira svetlobe od točke.

4.11 Razpršitev ozračja

Pri večjih razdaljah v odprtem prostoru pridejo do izraza učinki razpršitve svetlobe v ozračju (angl. *atmospheric scattering*). Rayleighovo sipanje (angl. *Rayleigh scattering*) je pojav, kjer se sončni žarki v ozračju elastično odbijejo od zračnih molekul in atomov, tj. delcev, veliko manjših od valovne dolžine svetlobe. Nebo je zaradi tega podnevi v celoti osvetljeno v predvsem modrih odtenkih, saj je razpršitev močno odvisna od valovne dolžine svetlobe [51].

Odboj žarkov od delcev večjih velikosti, kot nastane pri onesnaženem zraku, se imenuje Mie sipanje (angl. *Mie scattering*). Onesnažen zrak je pretežno sive barve, saj je pri velikih delcih razpršitev veliko manj odvisna od valovne dolžine svetlobe [44].

Sprememba kontrasta in odtenka oddaljenih predmetov je eden izmed večjih pokazateljev razdalje. Na sliki 7.12 je izražena simulacija razpršitve žarkov v ozračju.

Poglavje 5

Grabilnik

Za lažjo uporabo podatkov laserskega skeniranja Slovenije je bil razvit t. i. grabilnik virov (angl. *resource grabber*) – skripta, napisana za okolje *Node.js*. S skripto se prenese in pripravi podatke za uporabo v strežniku, ki je opisan v naslednjem poglavju.

5.1 Orodja

Grabilnik podatkov Slovenije je skripta, napisana za *Node.js* [43] JavaScript okolje s pomočjo razvojnega okolja Visual Studio Code [56]. Za pretvorbo oz. tako imenovano "osvoboditev" datotek se uporabi orodje *lasliberate* [23]. V prid lažjemu programiranju so bili uporabljeni naslednji Node.js moduli:

node-dbf Omogoči branje mreže listov iz dBase podatkovne baze [26].

async Za asinhrono obdelavo podatkov [35].

mustache Za uporabo predlog povezav [26].

mkdirp Zagotavlja obstoj direktorijev [18].

5.2 Definicije

V skripti so z robnimi ETRS89 koordinatami (poglavje 3.5.1) definirana vsa območja, ki naj jih skripta "pograbi" (koda 5.1). Med branjem listov so izključeni vsi kvadrati, katerih sredinska točka ni v vsaj enem izmed definiranih območij.

```
{
    name: "Ljubljana",
    min: { x: 457262.46, y: 96189.57 },
    max: { x: 467601.05, y: 106686.92 },
}
```

Koda 5.1: Primer definicije območja.

Definirani so tudi vsi tipi virov in njihove poti (koda 5.2).

Skripta deluje na principu obdelovalnih vrst knjižnice *async* [35]. V uporabi so vrste za

- vzpostavitev zapisov kvadratov `initRecordQueue`,
- vzpostavitev virov `initResourceQueue`,
- obdelavo virov `processResourceQueue`,
- zaključevanje virov `finishResourceQueue`,
- prenos `downloadQueue` in
- osvobajanje `liberatorQueue`.

5.3 Potek izvajanja

Prva akcija skripte je začetek branja dBase baze mreže listov cele Slovenije [40], v kateri je vsak kvadrat predstavljen z vrstico. Izmed stolpcev v bazi sta tudi stolpca `CENTERX` in `CENTERY`, ki predstavljata koordinate središčne točke

```
var lidarRemote = "http://gis.arso.gov.si/";
var lidarLocal = "W:/gis/arso/";
var liberator = lidarLocal + "lasliberate/bin/lasliberate.exe";

var configs = [
    {
        name: "lidar laz",
        source: {
            name: "lidar zlas",
            source:
                lidarRemote +
                "lidar/gkot/{{record.BLOK}}/D96TM/TM_{{record.NAME}}.zlas",
            drain:
                lidarLocal +
                "lidar/gkot/{{record.BLOK}}/D96TM/TM_{{record.NAME}}.zlas"
        },
        drain:
            lidarLocal +
            "laz/gkot/{{record.BLOK}}/D96TM/TM_{{record.NAME}}.laz",
        queue: liberatorQueue
    },
    {
        name: "map",
        source:
            lidarRemote +
            "arcgis/rest/services/opensource_dof84/MapServer/export" +
            "?bbox="+
            "{{bounds.min.x}},{{bounds.min.y}}," +
            "{{bounds.max.x}},{{bounds.max.y}}" +
            "&size={{image.width}},{{image.height}}" +
            "&bboxSR=3794&imageSR=3794" +
            "&format=png&f=image",
        drain:
            lidarLocal +
            "dof84/{{record.BLOK}}/{{record.NAME}}.png"
    }
]
```

name Ime tipa vira.

drain Predloga za izhodno pot datoteke na disku.

source Predloga za izvorni spletni naslov datoteke ali vgnezden podvir.

queue Vrsta obdelave (privzeta je vrsta za prenos).

Koda 5.2: Definicija virov in podvirov.

kvadrata. Vsi kvadrati, katerih središčne točke padejo v definirana območja (koda 5.1), se dodajo v vrsto `initRecordQueue`.

Vsaka vrsta lahko obdeluje le določeno število elementov naenkrat, in sicer 10 za `initRecordQueue`, `initResourceQueue`, `processResourceQueue` ter `finishResourceQueue`, ki samo obdelajo zahteve in jih postavijo v drugo vrsto, 2 za `downloadQueue` in 6 za `liberatorQueue`.

Potek obdelave posameznih vrst je opisan v sledečih sekcijah.

5.3.1 Vzpostavitev zapisa kvadrata (initRecordQueue)

Vse kombinacije zapisa iz vrste in tipov vira (koda 5.2) se dodajo v vrsto `initResourceQueue` kot nov vir. S tem se pripravi po en vir na vsako želeno izhodno datoteko.

5.3.2 Vzpostavitev vira (initResourceQueue)

Predloga za izhodno datoteko se preko podatkov v zapisu spremeni v konkretno pot (*izhod*). Če se definirana območja prekrivajo, se s tem lahko izloči podvojene vire. V primeru že obstoječe izhodne datoteke se vir doda v vrsto `finishResourceQueue`, saj ne potrebuje dodatne obdelave.

Če izhodna datoteka še ne obstaja, se s pomočjo knjižnice *mkdirp* [18] zagotovi, da najprej obstaja njen direktorij. Glede na definicijo izvorne datoteke sta dve možnosti. Če je definirana kot predloga za spletni naslov, se le-ta glede na zapis v obliki konkretnega naslova nastavi viru za *izvor* in doda v vrsto `processResourceQueue` za nadaljnjo obdelavo. Lahko pa je tudi definirana kot rezultat podvira. V tem primeru se v vrsto `initResourceQueue` doda nov vir z istim zapisom in podvirom kot tipom vira. Novemu viru se tudi določi, da je trenuten vir njegov starš.

5.3.3 Obdelava vira (processResourceQueue)

Vir se doda v vrsto, definirano v njegovem tipu. Privzeta je nalagalna vrsta `downloadQueue`. Če vir nima nastavljenega konkretnega izhodišča, je prišlo

do napake.

5.3.4 Zaključevanje vira (finishResourceQueue)

Viri brez starša ne potrebujejo dodatne obdelave. Pri virih s staršem pa je potrebno nadaljevati obdelavo. Starša se doda v vrsto `processResourceQueue`, za izvor pa se mu nastavi izhod vira (pot do izhodne datoteke). S tem se preda obdelavo vira v naslednji korak.

5.3.5 Prenos vira (downloadQueue)

Prične se prenos datoteke iz izvora vira v izhod vira, torej iz konkretnega spletnega naslova, zgrajenega s pomočjo predloge v koraku 5.3.2, v datoteko na disku. Ob uspešnem prenosu se vir doda v vrsto `finishResourceQueue`.

5.3.6 Osvoboditev vira (liberatorQueue)

LIDAR podatki na voljo za prenos so shranjeni v lastniškem stisnjenem formatu *zLAS*, zato se po prenosu za lažjo nadaljnjo uporabo datoteko pretvori oz. "osvobodi" v format *LAZ* s pomočjo orodja *lasliberate* [23]. Ob uspešni pretvorbi se vir doda v vrsto `finishResourceQueue`.

5.4 Prilagodljivost in odpornost

Obdelava z vrstami in definicijo virov in podvirov se izkaže kot prilagodljiv in odporen sistem. Prenesejo ali obdelajo se samo tisti viri, ki so potrebni za želene izhodne datoteke. Prenos in pretvorba med potekom obratujeta na začasnih datotekah, zato sistema v primeru prekinitve ne pustita v vmesnem stanju. Lena obdelava pa je tudi odporna na izbris vmesnih datotek, saj se osredotoča samo na končne produkte. Podvojene podatke v neželenih oblikah se tako lahko po obdelavi izbriše ali premakne – sistem jih ob naslednjih zagonih ne bo prenašal, dokler so želene izhodne datoteke še vedno na voljo.

Sistem temelji na obstoju datotek, zato je odporen tudi proti izbrisu ali pomanjkanju samo določenih datotek.

5.5 Primer

Manjkata ortofoto posnetka `457_101.png` in `457_102.png` ter LIDAR kvadrata v obliki LAZ `TM_457_98.laz` in `TM_457_99.laz`. Za `TM_457_98.laz` je že na voljo izvorna datoteka `TM_457_98.zlas`. Izvorna datoteka za `TM_457_99.laz` pa manjka, zato je potreben prenos s spletnega strežnika.

Koda 5.3 prikazuje izpis po zagonu skripte za omenjeni primer. Skripta najprej začne z branjem baze listov, iz katere doda vsak prebran zapis v vrsto `initRecordQueue`. Vsak zapis se nato kot kombinacija kvadrata in tipa virov doda v vrsto `initResourceQueue`. Najdeta se manjkajoči datoteki `457_101.png` in `457_102.png`, katerih prenos se takoj začne. S tem je vrsta `downloadQueue` zaradi omejitve hkratne obdelave zasičena. Manjkajoča datoteka `TM_457_98.laz` se doda v vrsto `liberatorQueue`, kjer se pretvori iz že obstoječe izvorne datoteke `TM_457_98.zlas`. Izvorna datoteka za manjkajočo datoteko `TM_457_99.laz` manjka, zato se najprej prenese s strežnika, nato pa še pretvori v želeno obliko. Ko so vse datoteke prenesene ter pretvorjene v pravo obliko, so vrste prazne in obdelava je zaključena.

```
>node grab-lidar.js
db parsing
db header parsed
  Ljubljana  map  457_101.png  download required
  Ljubljana  map  457_102.png  download required
  Ljubljana  map  457_101.png  downloading
  Ljubljana  map  457_102.png  downloading
  Ljubljana  lidar laz  TM_457_98.laz  liberation required
  Ljubljana  lidar laz  TM_457_98.laz  liberating
  Ljubljana  lidar zlas  TM_457_99.zlas  download required
db parsed
  Ljubljana  lidar laz  TM_457_98.laz  liberated
  Ljubljana  lidar zlas  TM_457_99.zlas  downloading
  Ljubljana  map  457_101.png  downloaded
  Ljubljana  map  457_102.png  downloaded
  Ljubljana  lidar zlas  TM_457_99.zlas  downloaded
  Ljubljana  lidar laz  TM_457_99.laz  liberation required
  Ljubljana  lidar laz  TM_457_99.laz  liberating
  Ljubljana  lidar laz  TM_457_99.laz  liberated
```

Koda 5.3: Izpis po zagonu skripte v opisanem primeru.

Poglavje 6

Strežnik

Lasersko zajeti (*LIDAR*) podatki Slovenije so zelo prostorsko zahtevni, zato je bil v namen obdelave in lažje integracije razvit strežnik v jeziku C++. Strežnik na zahtevo sledilnika iz primernih oblakov točk oz. kvadratov Slovenije zgradi kos sveta v obliki *škatle* za želen položaj in velikost. Ponuja tudi vpogled v stanje obdelave.

6.1 Orodja

Strežnik je napisan v jeziku C++ s pomočjo orodja za grajenje *CMake* [7], integriranega razvojnega okolja *Microsoft Visual Studio Community 2013/2015* [57] in naslednjih knjižnic:

LASlib Za branje LIDAR podatkov [22].

LASzip Podpora branju stisnjenih LIDAR podatkov [24].

civetweb Funkcionalnost spletnega strežnika [16].

nanoflann Hitro grajenje in iskanje k-d dreves [41].

DBFEngine Za branje dBase baze mreže listov [47].

amf-cpp V podporo serializaciji v AMF3[1] [55].

[1] Action Message Format.

ujson Za serializacijo v JSON[2] [27].

stb_image in stb_image_write Podpora branju in pisanju stisnjenih slik [4].

6.2 Arhitektura

Strežnik je zasnovan kot večnitni HTTP spletni strežnik. Streže več strani, njegova glavna naloga pa je, da za odjemalce generira in streže škatle v pravi obliki, velikosti in za pravi položaj. Osnovno delovanje večnitnega strežnika HTTP je podprto s strani knjižnice *civetweb* [16].

Ob prejemu zahteve za škatlo preko parametriziranega naslova 6.1 se naložijo primerni kvadrati oblakov točk. Škatla zahteve se nato zgradi preko korakov, opisanih v naslednjih sekcijah. Sliki 7.17 in 7.18 prikazujeta primerjavo korakov preko upodobitve sveta po vsakem koraku.

Za potrebe lažjega testiranja strežnik podpira stran z naslovom `/debug/`, na kateri je prikazana mreža ploščic tlorisov škatel. Vsaka ploščica je izrisana posebej, s čimer so izraženi robni pogoji in lokalne transformacije glede na škatlo. Slika 6.1 prikazuje primer mreže ploščic za Ljubljano.

6.3 Predpomnilnik

Ko strežnik prejme zahtevo za škatlo, najprej preveri predpomnilnik škatel. Implementiran je s prostorsko lokalno razpršeno tabelo, kar pomeni, da pride do sovpadanja indeksov le pri velikih razdaljah med bloki. Če je v predpomnilniku že škatla z enakimi parametri, se njene podatke le pošlje. V nasprotnem primeru se zgradi nova škatla in jo prepiše.

[2] JavaScript Object Notation.

Slika 6.1: Primer mreže ploščic Ljubljane na testni strani strežnika.

6.4 Nalaganje točk

Grajenje škatle se začne z nalaganjem oblaka točk iz ustreznih LAZ datotek. Knjižnica *LASlib* [22] s pomočjo *LASzip* [24] omogoča nalaganje točk v določenem mejnem pravokotniku X in Y koordinat. Kvadrati točk imajo dimenzije 1000 m × 1000 m, glede na višino Z pa so neomejeni. Za škatlo se preračuna koordinate in iz tega dobi ustrezni kvadrat točk za branje. Čeprav so škatle namenoma ponavadi veliko manjše od kvadratov, se potrebuje tudi do štiri kvadrate za škatlo, v primeru da prekriva njihovo robno mejo.

Po pridobitvi ustreznih kvadratov se iz vsakega preberejo točke, omejene oz. filtrirane z mejnim pravokotnikom škatle. Vmesnik za LASlib ne podpira uporabe iz več niti, zato se uporablja več bralcev oblakov točk za vsak kvadrat. Vsaka nit tako vzame prvega prostega bralca oz. počaka, da se kakšen sprosti. Nitna varnost je pri tem implementirana s pomočjo pogojnih spremenljivk, tj. `std::condition_variable`.

Iz enega ali več kvadratov se na koncu pridobi seznam točk, ki so znotraj meja grajene škatle. Točke vsebujejo lokacijo in klasifikacijo, ostale LIDAR specifične lastnosti pa se ne uporabijo.

`/gkot/box?x=`*x*`&y=`*y*`&z=`*z*`&sx=`*sx*`&sy=`*sy*`&sz=`*sz*
Parametri:

x Odmik škatle po osi x v blokih, npr. `896`.

y Odmik škatle po osi y v blokih, npr. `0`.

z Odmik škatle po osi z v blokih, npr. `-3072`.

sx Velikost škatle po osi x v blokih, npr. `64`.

sy Velikost škatle po osi y v blokih, npr. `256`.

sz Velikost škatle po osi z v blokih, npr. `64`.

Za zgornje primere je naslov zahteve naslednji:
`/gkot/box?sz=64&z=%2D3072&sy=256&y=0&sx=64&x=896`

Parametriziran naslov 6.1: Parametrizirana zahteva za škatlo.

6.5 Kvantizacija

Seznam točk p se nahaja v koordinatnem sistemu ETRS89 (glej 3.5.1), zato jih je potrebno pred vključitvijo v škatlo pretvoriti v koordinate blokov b. Operacija preslikave potrebuje referenčno točko r izhodišča škatle. Referenčno točko se pridobi s tem, da se z isto operacijo preslika koordinato ničtega bloka škatle, za referenčno točko r pa se vzame izhodišče koordinatnega sistema v ETRS89.

$$
\begin{aligned}
b &= [b_x, b_y, b_z] \\
d &= [d_x, d_y, d_z] \\
d &= p - r \\
b_x &= d_x \\
b_y &= d_z \\
b_z &= -d_y
\end{aligned}
$$

Vse točke vsebovane v bloku se preslikajo na iste koordinate bloka, zato pride do t. i. kvantizacije koordinat. Izključijo se vse točke, ki imajo koordinate bloka izven meja škatle, kar se lahko zgodi pri mejnih vrednostih.

V primeru več kot ene točke v bloku se za vrsto bloka uporabi zadnje nastavljena klasifikacija. Večjo natančnost se pridobi s štetjem vseh klasifikacij točk v posameznem bloku, kjer najbolj številna predstavlja boljši približek, vendar se to izkaže za potratno operacijo, razlika pa je minimalna.

6.6 Priprava podatkovnih struktur

Za lažjo nadaljnjo obdelavo se iz točk izračuna še nekaj podatkovnih struktur.

6.6.1 k-d drevesa

S pomočjo knjižnice *nanoflann* [41] se zgradi k-dimenzionalno drevo za vse točke, ki so vsebovane v škatli. Posebej se zgradi tudi k-d drevo vseh točk, klasificiranih kot tla. Ti strukturi bistveno pospešita dve pogosti operaciji v naslednjih korakih: iskanje bližnjih točk in iskanje točk v nekem radiju. Drevo vseh točk se izračuna šele po koraku zapolnitve tal (6.8.1). S tem se zajame tudi dodatne točke koraka.

6.6.2 Višine stolpcev

Za škatlo se izračuna višine vseh na tlorisu vidnih blokov. Enako se ponovi še za bloke, klasificirane kot tla. To pripomore poizvedbam višinske slike tal in struktur, ki se raztezajo nad njimi. Izračuna se tudi višina najnižjega in najvišjega nepraznega bloka.

6.7 Zapolnitev zgradb

V izvornih LIDAR podatkih so zgradbe v večini predstavljene samo s strehami, podatki sten pa manjkajo. Privzame se, da imajo vse zgradbe navpično

ravne stene. Za vsak blok, ki je klasificiran kot zgradba, se proti najnižji višini spremeni bloke v klasifikacijo zgradbe. S tem se vse strehe zgradb uporabijo kot meje za stene, kar je za večino zgradb dober približek.

6.8 Zapolnitev tal

Točke LIDAR podatkov nepopolno prekrivajo objekte in površine, zato so ponekod v tleh luknje. Prav tako pa so točke definirane samo na površini.

Za vse talne stolpce, izračunane v koraku 6.6.2, se predvideva, da imajo tla določena, če so višja od 0. Ostali stolpci so luknje v tleh.

6.8.1 Iterativno iskanje višine

Za vsako luknjo se izvede iterativno iskanje višine. V prvem koraku se nastavi točko iskanja na središče bloka na dnu stolpca luknje. V naslednjem koraku se poišče najbližjo talno točko iskalni točki s pomočjo k-d drevesa tal, izračunanega v koraku 6.6.1. Višino iskalne točke se nastavi na višino najdene talne točke in iskanje se ponovi. Iteracija poteka, dokler se višina spreminja za več, kot je določeno (npr. 0.5 m). Iskalna točka se nato spremeni nazaj v koordinate bloka. Višina bloka se nastavi kot nov približek višine tal v stolpce tal.

6.8.2 Polnitev

Za vsak stolpec se od najnižjega bloka do višine tal nastavi klasifikacija bloka na klasifikacijo tal. S tem se zapolni vse luknje in bloke pod tlemi.

6.9 Specializacija

Klasifikacija izvirnih podatkov je omejena, vendar do določene meje zanesljiva. Originalne klasifikacije so sledeče: [25]

Vrednost	Pomen
0	ustvarjeno, nikoli klasificirano
1	neklasificirano
2	tla
3	nizka vegetacija (do 1 m)
4	srednja vegetacija (1 m – 3 m)
5	visoka vegetacija (nad 3 m)
6	zgradba
7	šum

Za bolj točno predstavitev značilnosti okolja se klasifikacije najvišjih blokov (blokov, vidnih v tlorisu) specializira s pomočjo ortofoto posnetkov, predstavljenih v sekciji 3.5.2. Dodatne klasifikacije so definirane glede na standardne vrednosti, predpisane s strani *ASPRS*[3] in dovoljene lastne vrednosti:

Vrednost	Pomen
9	voda
70	oranžni strešniki
71	sivi strežniki
75	ploščata streha
80	asfalt
81	beton

Prvi korak specializacije bloka je preslikava koordinat v koordinatni sistem ETRS89. LIDAR kvadrati so usklajeni z ortofoto posnetki, zato se preko teh koordinat le uporabi posnetek že naloženega kvadrata. Koordinate se preslika še v koordinatni sistem posnetka, iz katerega se prebere barvno vrednost piksla. Glede na originalno klasifikacijo in na vrednost piksla se nato blok specializira v bolj specifično klasifikacijo.

Vsaka originalna klasifikacija se lahko preslika samo v vnaprej določene klasifikacije:

[3]American Society for Photogrammetry & Remote Sensing [29].

Originalna klasifikacija	Specializirane klasifikacije
tla	voda, nizka vegetacija, asfalt, beton
nizka vegetacija	voda, nizka vegetacija, asfalt, beton
srednja vegetacija	srednja vegetacija
visoka vegetacija	visoka vegetacija
zgradba	oranžni strešniki, sivi strežniki, ploščata streha
šum	šum

Preslikave so bile izbrane empirično. Večino cest je bilo klasificiranih kot tla ali nizka vegetacija, zato se te preslikajo v asfalt ali beton. Strehe zgradb se preslikajo v različne vrste, saj imajo velik vpliv na izgled mesta iz ptičje perspektive. Izbira med določenimi specializiranimi klasifikacijami se naredi s pomočjo preslikovalne tabele barvnih vrednosti 6.1. Za vrednosti piksla p se med možnimi klasifikacijami s izbere tisto, ki ima najmanjšo evklidsko razdaljo r oz. r^2 v RGB barvnem prostoru, kot prikazujejo sledeče enačbe:

$$
\begin{aligned}
p &= [p_r, p_g, p_b] \\
s &= [s_r, s_g, s_b] \\
d &= [d_r, d_g, d_b] \\
d &= s - p \\
r^2 &= d_r^2 + d_g^2 + d_b^2
\end{aligned}
$$

Sistem ima za določene originalne klasifikacije tudi podporo branja več specializacij okoli bloka, pri katerem se vzame najbolj številna. To pripomore k manj šumnim rezultatom.

6.10 Filtriranje

Po prejšnjih korakih imajo podatki še veliko šuma zaradi nenatančne specializacije in nepopolnih originalnih podatkov. S filtriranjem se zmanjša nepravilnosti in izboljša izgled. Postopek se izvaja s pomočjo več filtrov. Filtri so

Barva	Specializirana klasifikacija	Barva	Specializirana klasifikacija
1a3f40	voda	5f666b	sivi strešniki
404943	voda	5f666b	sivi strešniki
5d7846	visoka vegetacija	858889	sivi strešniki
35502e	visoka vegetacija	858889	sivi strešniki
262127	visoka vegetacija	787170	sivi strešniki
5f7056	nizka vegetacija	8a8787	ploščata streha
567152	nizka vegetacija	a7a3a1	ploščata streha
636e51	nizka vegetacija	b2aeb0	ploščata streha
435e3d	nizka vegetacija	dedcdc	ploščata streha
334a2f	nizka vegetacija	949293	asfalt
c7a08c	oranžni strešniki	858586	asfalt
9d695d	oranžni strešniki	969392	asfalt
725251	oranžni strešniki	818081	asfalt
c39583	oranžni strešniki	858586	asfalt
c1937f	oranžni strešniki	a3a1a2	asfalt
ae7264	oranžni strešniki	aaa6a5	asfalt
b68978	oranžni strešniki	cac5c7	beton
d0a794	oranžni strešniki	d0cacb	beton
bda89d	oranžni strešniki	bfbbba	beton

Klasifikacije senčnih predelov

Barva	Specializirana klasifikacija
040814	visoka vegetacija
273649	beton
253748	beton
38414e	asfalt
1b2b3e	asfalt
0f1e30	nizka vegetacija
060b25	voda
0e2231	voda
132e38	voda

Tabela 6.1: Preslikovalna tabela barvnih vrednosti v specializirane klasifikacije. Barve so definirane kot šestnajstiško število `RRGGBB`, kjer so `RR` rdeča, `GG` zelena in `BB` modra barvna komponenta v razponu 0–255.

predstavljeni v tabeli 6.2.

6.11 Izenačevanje vodne površine

Vodna površina je zaradi šumnih podatkov na nekaterih delih, še posebno ob rečnem bregu in ob mostovih, lahko nagrbančena in neprepričljiva. Za škatlo se tako najprej poišče višine vseh vodnih blokov, nato pa izračuna mediano in to določi kot enotno vodno površino v škatli. Vse vodne bloke površine se popravi na izračunano vodno višino.

6.12 Poglabljanje vodne povšine

Voda je specializirana samo na površini, zato ni pravih podatkov o globini. Vodna telesa v naravi ponavadi niso plitva, zato se vodne površine umetno poglobi. Vodna globina je odvisna od oddaljenosti bloka vodne površine do kopnega. Oddaljenost se ugotovi s spiralnim iskanjem okoli bloka. V višini vodne površine škatle se iterativno preverja tip vedno bolj oddaljenih blokov v vse štiri smeri neba do določene meje. Približna oddaljenost kopnega je tako oddaljenost do prvega najdenega ne-vodnega bloka. Razdaljo se vzame kot globino, vse bloke pod blokom vodne površine pa se spremeni v vodo do določene globine.

6.13 Pretvorba klasifikacij

Bloki in klasifikacije so pripravljeni, zato jih je potrebno še pretvoriti v tipe blokov, ki jih podpira sledilnik žarkov. Vsaka klasifikacija se preslika v tip bloka sledilnika žarkov preko preslikovalne tabele 6.3.

Ciljna klasifikacija	Viri	Radij	Mejno razmerje
nikoli klasificirano	neklasificirano	$\sqrt{2}$	-1
asfalt	nizka vegetacija, srednja vegetacija, tla, beton	2	-0.2
oranžni strešniki	ploščata streha, sivi strešniki	3	0.2
ploščata streha	oranžni strešniki	3	-0.1
sivi strešniki	oranžni strešniki	3	-0.1
nikoli klasificirano	ploščata streha	4	0.5
voda	nizka vegetacija, srednja vegetacija, asfalt, beton	6	0.1

Parametri:

ciljna klasifikacija Klasifikacija, v katero se blok spremeni ob izpolnitvi pogojev. Če je ta nastavljena na nikoli klasificirano, se za klasifikacijo privzame najbližja bloku.

viri Klasifikacije, ki se lahko preslikajo v ciljno klasifikacijo.

radij Radij okoli bloka, v katerem se išče vire.

mejno razmerje Kontrolira možnost preslikave. Na primer, ciljna klasifikacija se uporabi pri vrednosti:

- 1, če imajo vse točke v radiju ciljno klasifikacijo,
- 0.5, če ima vsaj 50% točk v radiju ciljno klasifikacijo,
- 0.25, če ima vsaj 25% točk v radiju ciljno klasifikacijo,
- 0, če ima vsaj pol točk ciljno klasifikacio in pol točk klasifikacijo vira,
- -0.5, če ima manj kot 50% točk v radiju klasifikacijo vira,
- -1, če ima manj kot 100% točk v radiju klasifikacijo vira.

Tabela 6.2: Filtri blokov in njihovi parametri.

Klasifikacija	Tip bloka
nikoli klasificirano	zrak
neklasificirano	zrak
tla	zemlja
nizka vegetacija	trava
srednja vegetacija	listje smreke
visoka vegetacija	listje hrasta
zgradba	svetlo siva volna
šum	zrak
voda	voda
oranžni strešniki	oranžna volna
sivi strešniki	svetlo siva volna
ploščata streha	bela volna
asfalt	kamen
beton	svetlo siva volna

Tabela 6.3: Preslikovalna tabela klasifikacije v tip bloka sledilnika žarkov. Vrednosti so bile izbrane empirično za podoben izgled izvirnim materialom.

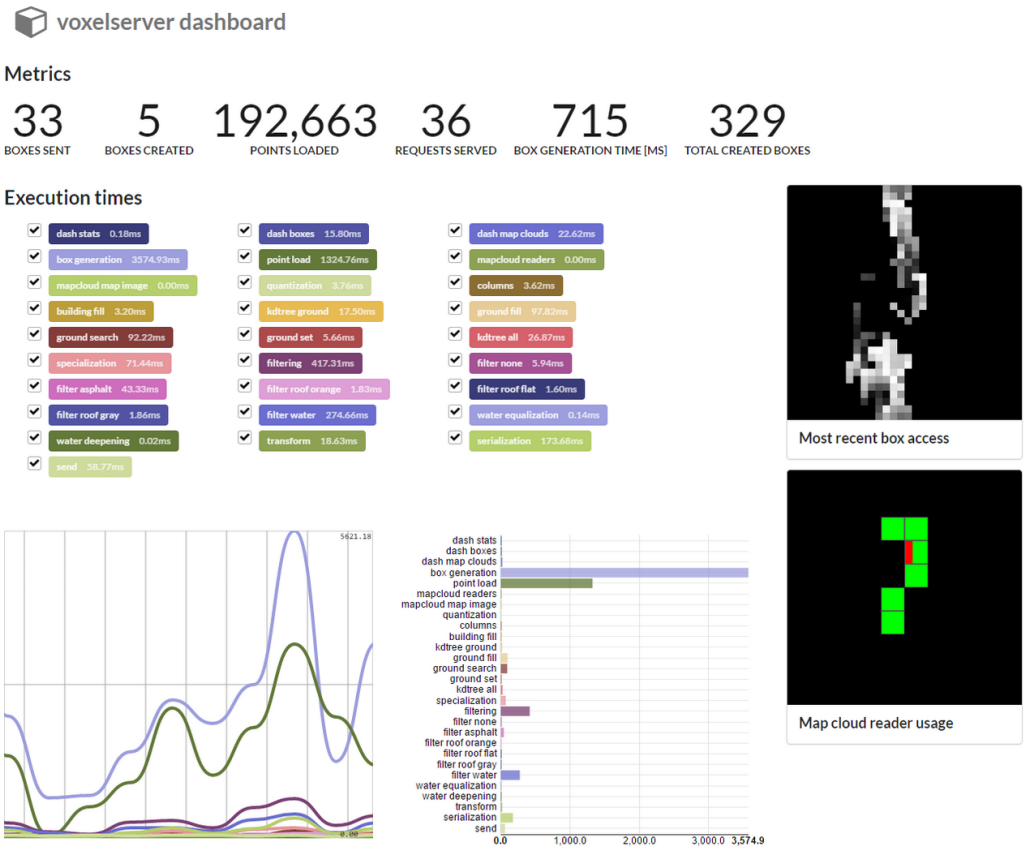
6.14 Serializacija in pošiljanje

Obdelani bloki se s pomočjo knjižnice *amf-cpp* [55] serializirajo v AMF[4] obliko primerno branju s strani sledilnika žarkov. Serializirana oblika škatle se shrani v predpomnilnik škatel za nadaljnjo uporabo in pošlje odjemalcu. S tem se obdelava škatle zaključi, nit strežnika pa je pripravljena na naslednjo zahtevo.

6.15 Nadzorna plošča

Za potrebe vpogleda v stanje strežnika med izvajanjem je bila razvita stran z nadzorno ploščo, prikazana na sliki 6.2. Ponuja hiter pregled internega stanja obdelave škatel, dostopa predpomnilnika, bralcev LIDAR podatkov in izvajalnega časa različnih korakov obdelave. Za hitrejši razvoj so bile uporabljene naslednje knjižnice:

[4] Action Message Format.



Slika 6.2: Nadzorna plošča strežnika med upodabljanjem prizora z večjim številom škatel.

Semantic UI Za enostaven in hiter razvoj uporabniškega vmesnika [33].

D3.js Za vezavo podatkov na elemente [6].

Smoothie Charts Prikaže izvajalni čas korakov skozi čas [59].

NVD3 Podpora različnih grafikonov za *D3.js* [39].

jQuery Omogoča poenostavljen dostop do elementov strani [49].

Poglavje 7

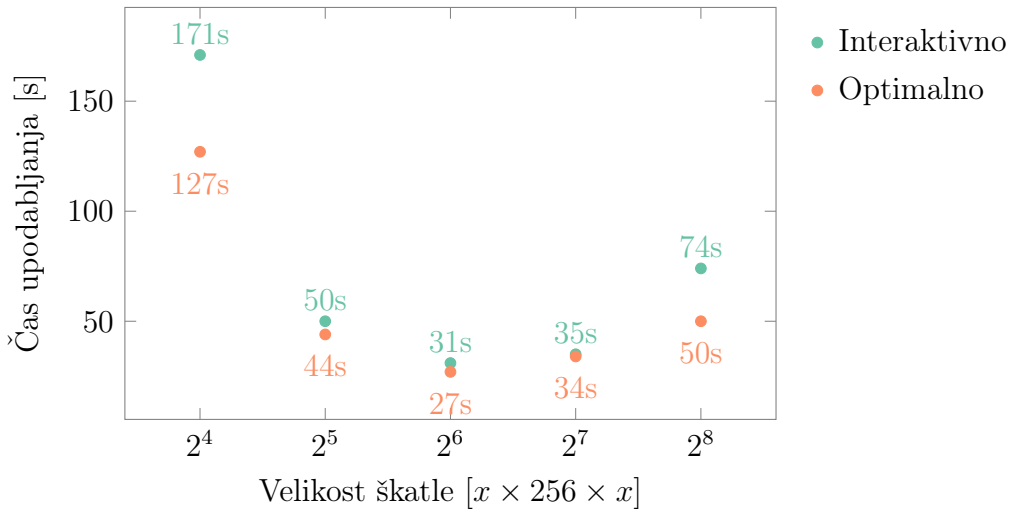
Rezultati

7.1 Raziskava parametrov

Testiranje je potekalo na sistemu s procesorjem *Intel Core i7 5820K* s šestimi jedri na taktu 3.4 GHz, 32 GiB 2400 MHz pomnilnikom in trdim diskom *Samsung SSD 850 Pro 512GB*. Za pogled je bila nastavljena ≈ 500 m visoka ptičja perspektiva Ljubljane s 40° vidnim poljem. Kjer ni omenjeno drugače, je obdelovalo 12 sledilnikov preko vtičnice z omejitvijo pomnilnika 700 MiB na sledilnik in velikostjo škatle $64 \times 256 \times 64$. Zahtevnost upodobitve je zelo odvisna od pogleda, ker so bližnji pogledi lahko veliko hitrejši zaradi lokalnosti podatkov in polnih predpomnilnikov, oddaljeni pa počasnejši zaradi večjega števila škatel, manjše lokalnosti in v splošnem več podatkov. Omenjeni pogled je srednje oddaljen, vsebuje pa toliko škatel, da jih sledilnik ne more imeti naenkrat naloženih v 700 MiB pomnilnika.

7.1.1 Velikost škatle

Preizkušene so bile različne velikosti škatle. Odvisnost časa upodobitve od velikosti je prikazana na sliki 7.1. Za določen sistem in pogled je bila ugotovljena najbolj optimalna velikost 64×64.



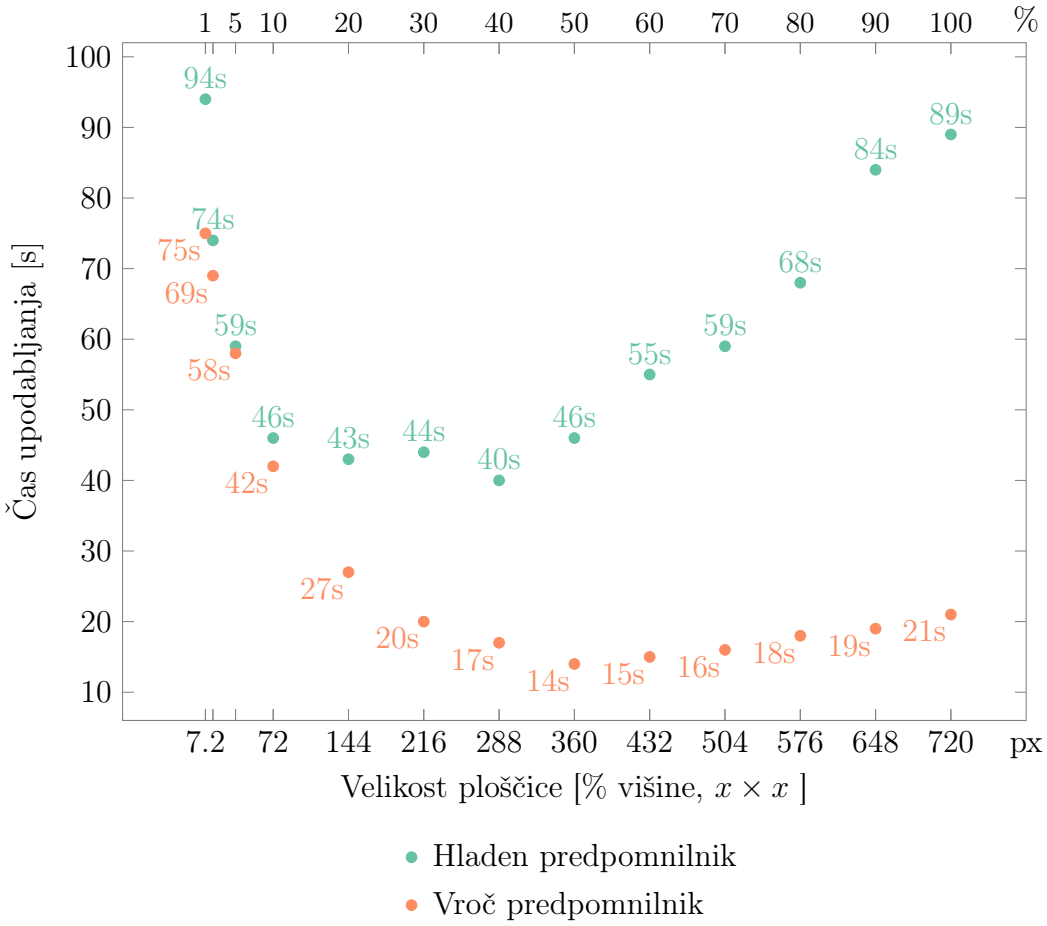
Slika 7.1: Čas upodabljanja v odvisnosti od velikosti škatle 16–256. Velikosti so omejene na potence števila 2. Opravljeni sta bili dve izvedbi — interaktivna (več ločljivosti, spiralni vrstni red ploščic) in optimalna (samo končna ločljivost, navpičen vrstni red ploščic). Najbolj optimalna velikost na testiranem sistemu je 64.

7.1.2 Velikost ploščice

Preizkušene so bile tudi različne velikosti ploščic. Odvisnost časa upodobitve od velikosti je prikazana na sliki 7.2. Velikost ploščice je pri danih nastavitvah in vročem predpomnilniku optimalna pri 50% višine slike. Tako velike ploščice pa so sicer pomnilniško zahtevne, težko je pa tudi sprotno spremljati potek. Obnašanje se spremeni pri manj idealnih pogojih hladnega predpomnilnika, kjer se zaradi zahtev škatel čas upodabljanja ustali pri velikosti 10% – 40%, nato pa zaradi manjšega števila sledilnikov začne rasti. Najboljši kompromis je velikost ploščice $\approx$ 100px oz. $\approx$ 15% višine slike.

7.1.3 Nastavitve upodabljanja

Izmerjen je bil vpliv nekaterih nastavitev na čas trajanja upodobitve. Pred vsako upodobitvijo je bil spraznjen predpomnilnik sledilnikov, škatle pa so bile že prednaložene na strežniku. Po vsaki upodobitvi je bilo opravljeno kontrolno trajanje zelo preprostega upodabljanja brez dodatnih nastavitev.



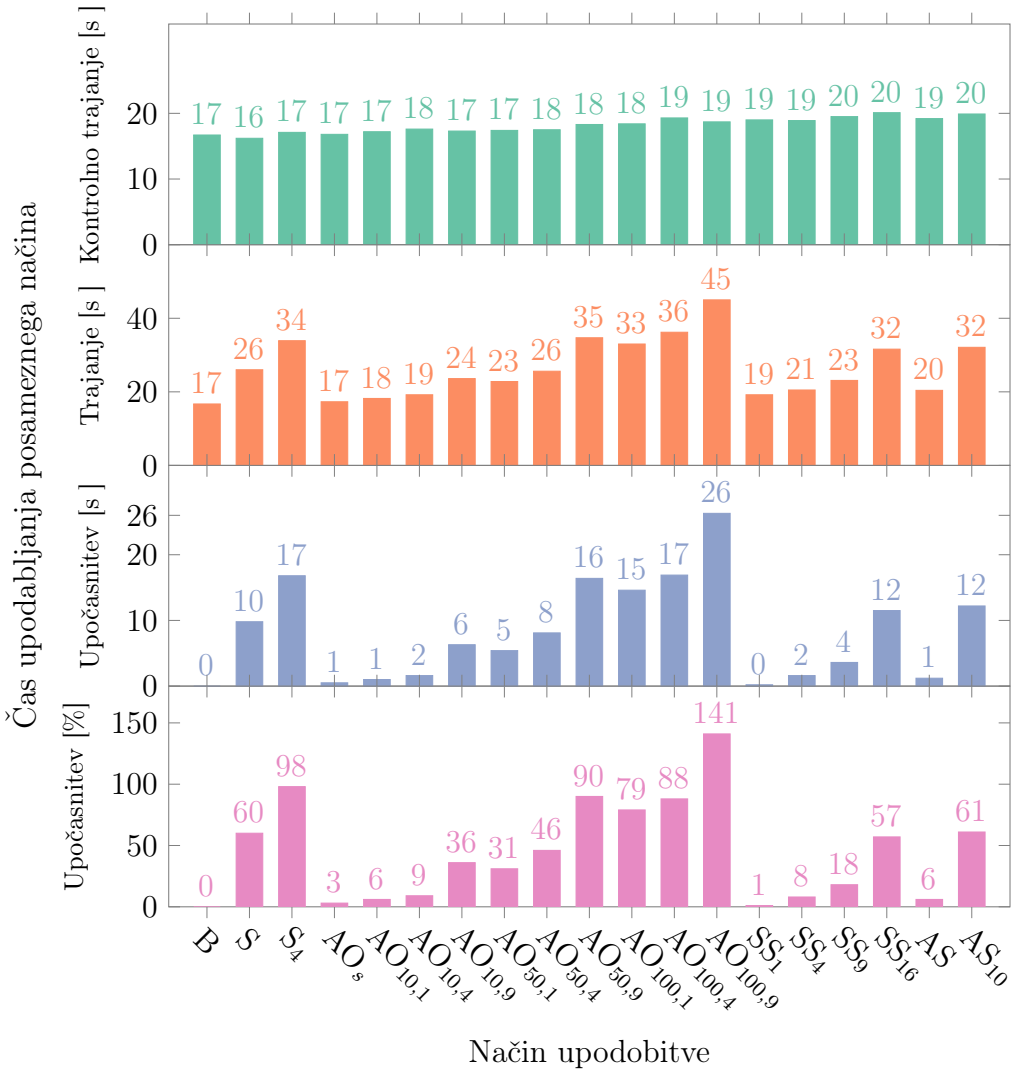
Slika 7.2: Čas upodabljanja v odvisnosti od velikosti ploščice. Dimenzije slike so 1280×720 pikslov. Velikost ploščice je izražena tako v procentu višine kot v pikslih. Prikazani so časi dveh izvedb: z vročim (vse škatle so že pred-naložene) in s hladnim predpomnilnikom strežnika (takoj po zagonu strežnika).

Kontrolna trajanja, absolutni časi in upočasnitev so prikazani na sliki 7.3. Upočasnitev je definirana kot razlika med absolutnim časom trajanja upodobitve in kontrolnim trajanjem. Relativna upočasnitev v odstotkih je definirana kot procentualna upočasnitev upodobitve od kontrole. Preizkušene so bile nastavitve sence (navadne in mehke s štirimi žarki), okoliškega senčenja (preprostega in z enim, štirimi in devetimi žarki v kombinaciji z radijem 10, 50 in 100), supervzorčenja (angl. *supersampling*, 1, 4, 9 ali 16 vzorcev na piksel) in razpršitev ozračja (navadna ali volumetrična z desetimi vzorci na poti).

Sence imajo konkreten vpliv na čas upodobitve, mehke sence pa so kljub štirikratnemu številu senčilnih žarkov samo $98/60 \approx 63\%$ počasnejše od ostrih. Kot pričakovano, preprosto okoliško senčenje nima velikega vpliva na čas upodobitve. Okoliško senčenje radija 10 in štirih žarkov doseže podobno zahtevnost z boljšim izgledom. Višje kvalitete okoliškega senčenja imajo veliko večjo zahtevnost, ki je v večini odvisna od radija. Še ena prelomna točka zahtevnosti pa je med štirimi in devetimi žarki. Supervzorčenje ima manjši vpliv, kot bi bilo sprva pričakovano, vendar je v osnovi veliko časa porabljenega za prenos škatel, dodatno pa so v igri tudi učinki lokalnosti žarkov. Razpršitev ozračja v osnovi ne vpliva preveč na čas upodobitve, je pa volumetričen način z desetimi senčilnimi žarki v rangu zahtevnosti navadnih senc.

7.1.4 Vrsta obdelovalcev in predpomnilnik

Vpliv vrste obdelovalca in velikosti predpomnilnika je prikazan na sliki 7.4. Pred prvim zagonom vsake velikosti je bil predpomnilnik spraznjen, nato je bilo opravljenih 10 zaporednih zagonov. S tem se meri izrabljenost predpomnilnika skozi več zagonov. Primerja se tudi različne vrste obdelovalcev. Zeleni obdelovalec lahko v celoti izrablja določeno velikost predpomnilnika, delavci pa si mejo medsebojno delijo. Preizkušena sta bila samo dva istočasna delavca, saj je posamezen program v okolju Flash Player oz. AIR ponavadi stabilen samo do $\approx 1.2\,\mathrm{GiB}$ pomnilnika, kar pomeni, da je z večjim



Oznaka	Ime	Oznaka	Ime
B	Kontrola	$AO_{100,1}$	O. s. razpon 100 — 1×1 žarek
S	Sence	$AO_{100,4}$	O. s. razpon 100 — 2×2 žarka
S_4	Mehke sence — 2×2 žarka	$AO_{100,9}$	O. s. razpon 100 — 3×3 žarki
AO_s	Preprosto okoliško senčenje	SS_1	Supervzorčenje — 1×1
$AO_{10,1}$	O. s. razpon 10 — 1×1 žarek	SS_4	Supervzorčenje — 2×2
$AO_{10,4}$	O. s. razpon 10 — 2×2 žarka	SS_9	Supervzorčenje — 3×3
$AO_{10,9}$	O. s. razpon 10 — 3×3 žarki	SS_{16}	Supervzorčenje — 4×4
$AO_{50,1}$	O. s. razpon 50 — 1×1 žarek	AS	Razpršitev ozračja
$AO_{50,4}$	O. s. razpon 50 — 2×2 žarka	AS_{10}	Volumetrična razpršitev ozračja
$AO_{50,9}$	O. s. razpon 50 — 3×3 žarki		

Slika 7.3: Primerjava časa upodobitve različnih načinov upodabljanja.

številom delavcev večja prostorska stiska. Z nekoliko večjo zahtevo izbranega pogleda to pomeni, da zaradi delitve več delavcev nima dovolj pomnilnika za pravilno obratovanje. Meja pri podprogramskih obdelovalcih preko vtičnice velja za vsakega posebej. Upodabljanje je bilo preizkušeno z enim, dvema, štirimi, šestimi in dvanajstimi podprogrami preko vtičnice. Nastavljene so bile najbolj preproste nastavitve sledilnika za najvišjo hitrost, razen pri 12 zahtevnih vtičnicah, kjer so bile vklopljene sence, razpršitev ozračja, okoliško senčenje in supervzorčenje.

Iz rezultatov sledi, da je 300 MiB pomnilnika občutno premalo za izbran pogled ne glede na vrsto obdelovalca, saj preko več zagonov predpomnilnik ne pripomore k hitrosti upodabljanja. Še več, z večimi sledilniki se sistem upočasni, najverjetneje zaradi povečanega prometa škatel med sledilniki in strežnikom. Pri omejitvi 600 MiB je že opazna razlika med obdelovalci. Pri zelenem obdelovalcu ta podvojitev ne spremeni hitrosti, kar pomeni, da ima še vedno premalo prostora za vse škatle pogleda. Prav tako velja za enega in dva podprograma preko vtičnice. Delavca imata manjšo prostorsko stisko, zato se približata zelenemu obdelovalcu. Štirje podprogrami so v prvem zagonu enako hitri kot v prejšnjem koraku, v naslednjih pa se zaradi efekta predpomnilnika približajo zelenemu obdelovalcu. Tako se obnaša tudi šest in dvanajst podprogramov, le da v naslednjih zagonih krepko presežejo hitrost zelenega obdelovalca.

Omejitev 900 MiB dodatno poudari vrednost predpomnilnika, ker se škatle v njem menjavajo veliko redkeje. Vrste z več kot enim podprogramom se konkretno pohitrijo skozi zaporedne zagone. Ostale imajo še vedno pomanjkanje pomnilnika. Ob tem koraku se tudi vidi relativna razlika med zahtevnim in hitrim upodabljanjem preko dvanajstih programov. Hitra upodobitev je omejena s prenosom in nalaganjem škatel, zato je, zaradi polnjenja predpomnilnikov posameznih podprogramov, vedno hitrejša z vsakim zagonom. Zahtevna upodobitev pa se pohitri samo do neke mere, saj ob določeni točki postane omejena s strani sledenja velikega števila žarkov skozi voksle.

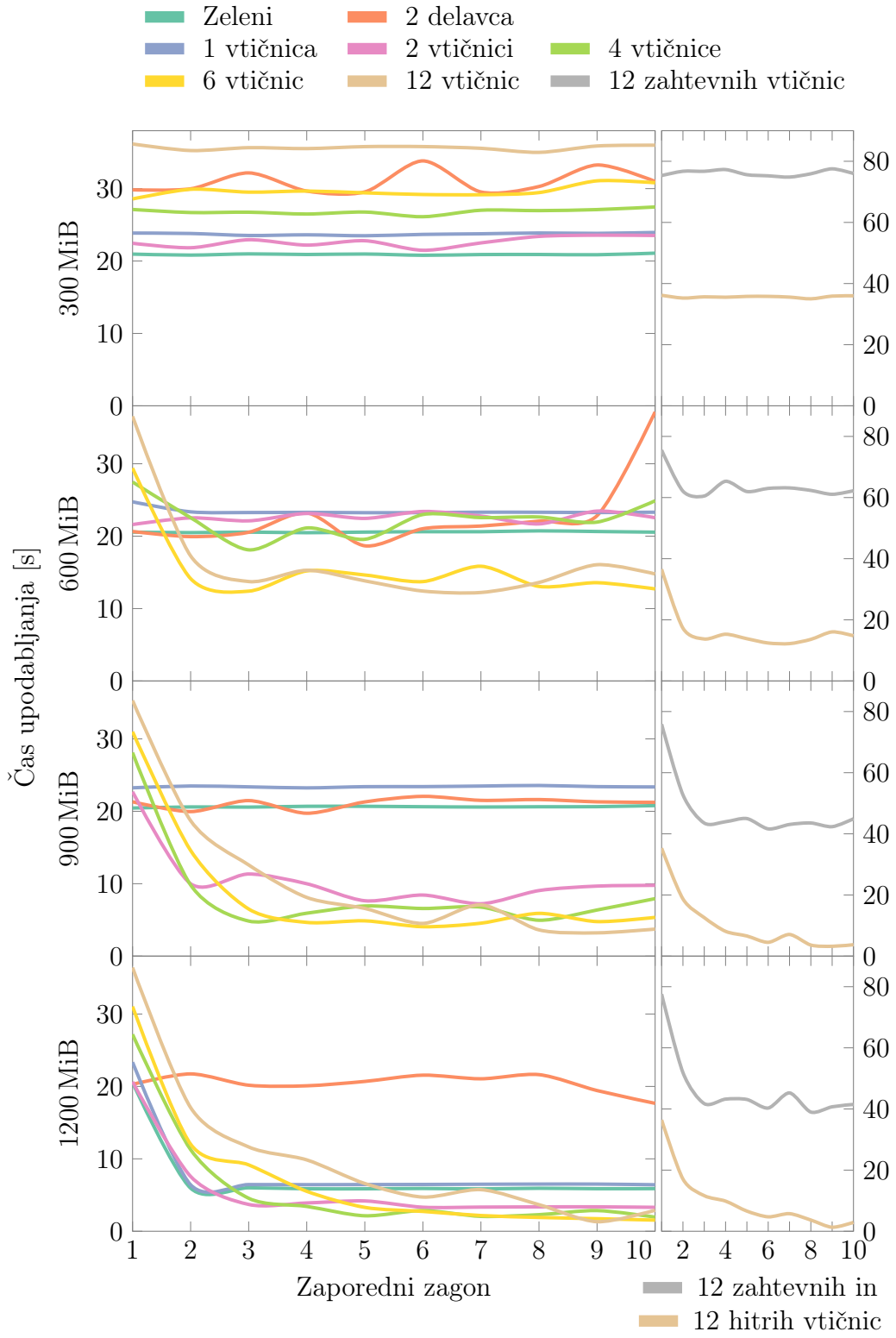
Najvišja preizkušena omejitev 1200 MiB ima še nekaj dodatnih zanimivo-

sti. Opazno delavca za veliko število škatel nista primerna, saj si pomnilniški prostor delita. Zaradi tega imata tudi pri tej omejitvi glede na zagon konstantno hitrost. Poleg ostalih se po prvem zagonu pohitrita še zeleni obdelovalec in en podprogram preko vtičnice. Po prvem zagonu imata, ker sta en sledilnik, pomnjene vse škatle, zato se hitrost po drugem zagonu ne spreminja več. Pri več vtičnicah pa različni podprogrami skozi zagone nalagajo različne predele škatel pogleda, zato se predpomnilnik posameznih podprogramov počasi polni, s tem pa se viša tudi hitrost. Iz tega se opazi kompromis med hitro "končno hitrostjo" in hitrim "pospeškom" oz. odzivom predpomnilnikov. Konkretno, za število podprogramov, je odziv samo z enim zelo hiter, saj ima že po prvem nalaganju vse škatle v pomnilniku in tako doseže svojo najvišjo hitrost, ki pa se ne spreminja v naslednjih korakih. Dva podprograma sta v drugem zagonu še počasnejša od enega podprograma, v tretjem pa hitrejša. Pri najvišjem številu testiranih podprogramov (12) je bila po desetih zagonih dosežena najvišja hitrost.

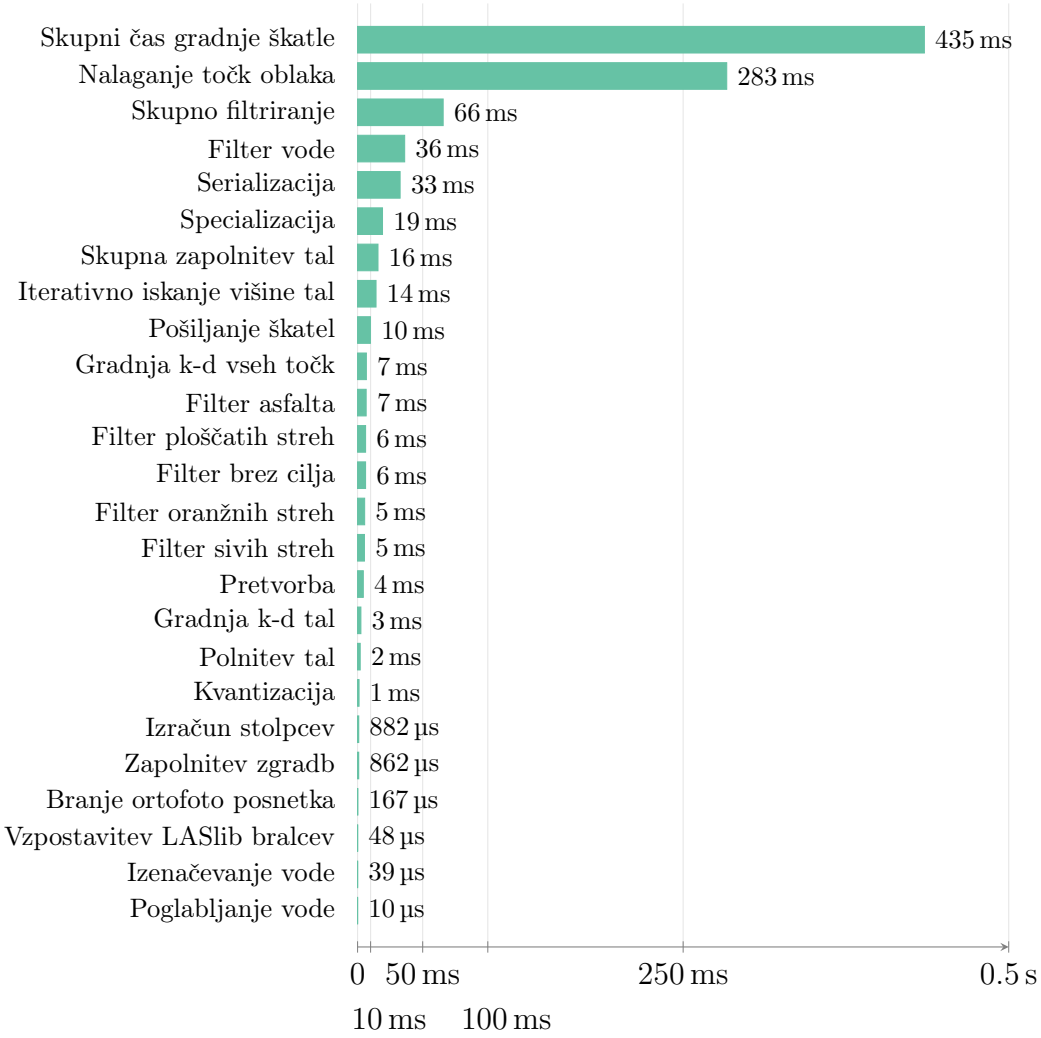
Ta efekt je v večini odvisen od pogleda in zahtevnosti, zato je izbira vrste obdelovalcev v večini odvisna od nastavitev in uporabe. Za bolj interaktivno uporabo in hitro premikanje je primeren zeleni obdelovalec ali manjše število podprogramov, za učinkovito in hitro upodobitev zahtevnih pogledov pa je bolj primerno večje število podprogramov.

7.2 Izvajanje strežnika škatel

Izmerjeni so bili časi trajanja posameznih korakov strežnika škatel pri upodabljanju pogleda, opisanega v poglavju 7.1. Strežnik je imel na začetku prazen predpomnilnik škatel, saj bi se pri že naloženih škatlah merilo le korak pošiljanja. Upodabljanje oz. gradnja škatel je trajala slabe pol minute. Vsako sekundo se je pridobil vzorec trajanja korakov v prejšnji sekundi. Strežnik ima več niti, zato se lahko gradi več škatel istočasno. Trajanje je s tem seštevek vseh niti, iz tega pa sledi, da obdelava lahko traja več sekund vsako sekundo.



Slika 7.4: Odvisnost različnih vrst upodobiteljev od predpomnilnika različnih velikosti.



Slika 7.5: Povprečno trajanje korakov posamezne škatle.

Povprečno trajanje korakov vseh 284 naloženih škatel je prikazano na sliki 7.5. Nekateri koraki so sestavljeni iz drugih korakov, zato so označeni, da predstavljajo "skupno" trajanje. Nalaganje točk z diska vzame daleč največ časa. Večino časa porabi knjižnica LASlib, ki najprej točke išče v indeksu, nato pa jih mora razširiti iz stisnjene oblike, kar je časovno zahtevno. Filtriranje je naslednji najbolj zahteven korak, v večini zaradi velikega radija pri vodnem filtru. Sledijo serializacija, specializacija in pošiljanje škatel, naslednji koraki pa so krajši od 10 ms.

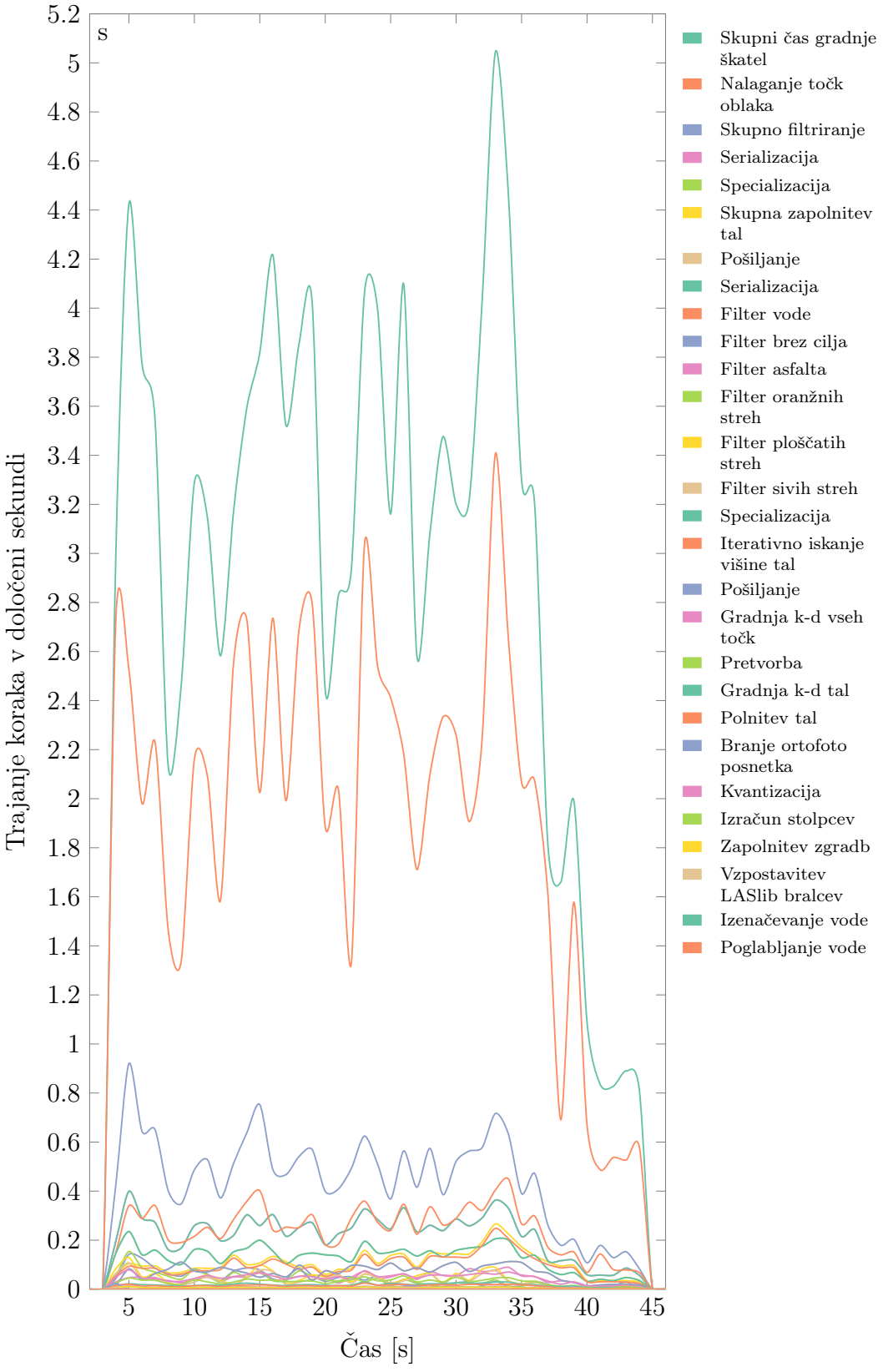
Slika 7.6 prikazuje pregled trajanja korakov skozi čas. Vsako sekundo se zgradi in pošlje po več škatel iz različnih niti, s tem se gradi po več sekund vsako sekundo. Mediana gradnje škatel je 3 s, kar pomeni, da pri danih nastavitvah strežnik najbolj pogosto zasede okoli 3 jedra procesorja.

Trajanja so bolj podrobno glede na velikostni red in namen prikazana na sliki 7.7. Prvi graf prikazuje daljše korake v primerjavi z nekaj ostalimi na logaritemski lestvici. Drugi graf prikazuje serializacijo in vse filtre. Vidi se, da je trajanje obdelave filtra v večini odvisno od njegovega radija, saj je filter vode radija 6 m pri ≈ 300 ms najzahtevnejši, ostali filtri radija okoli 3 m pa približno isto zahtevni pri ≈ 50 ms.

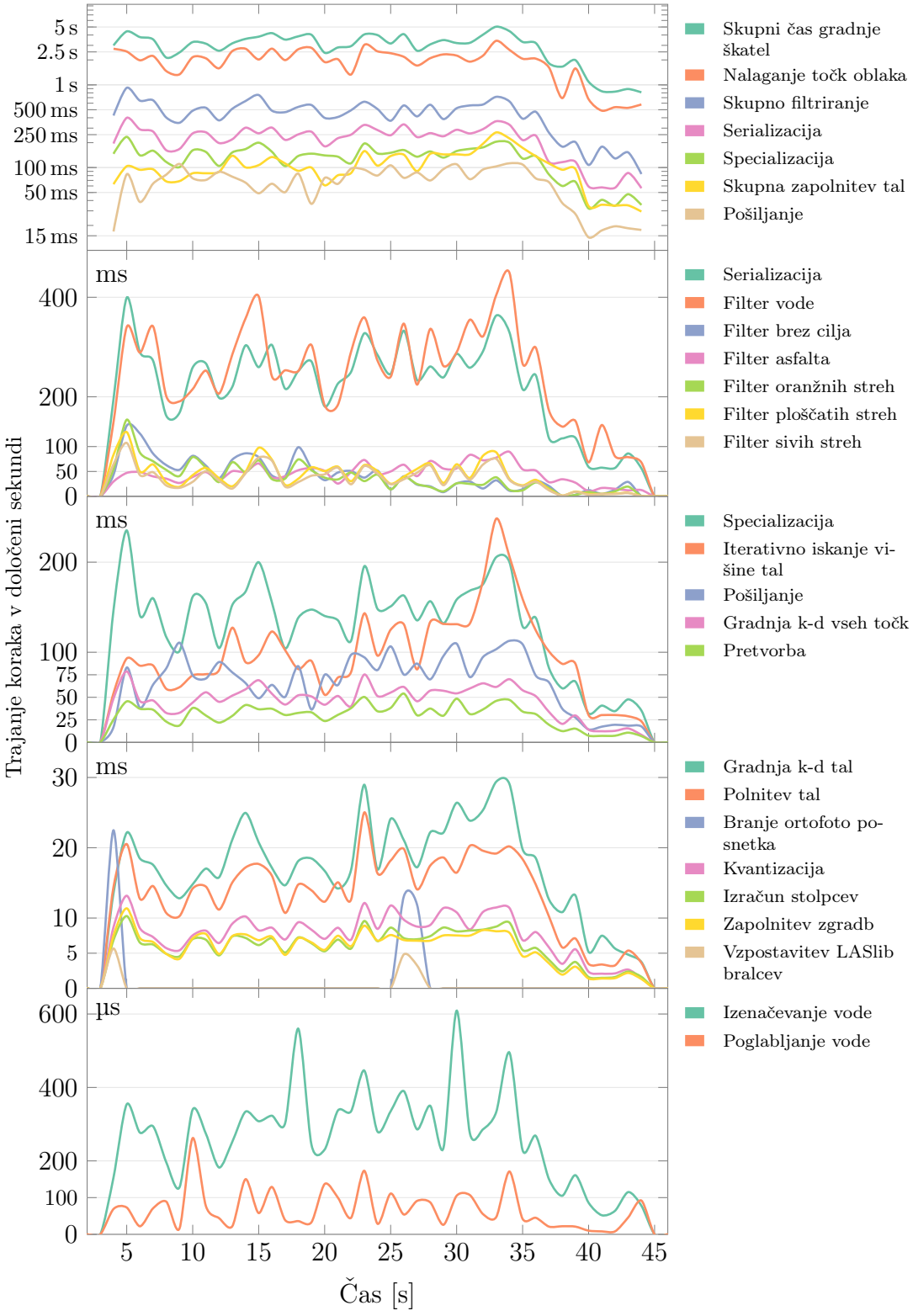
Sledi graf nekaterih srednje zahtevnih korakov. Specializacija je zaradi pravil in branja slike dokaj zahteven proces. Iterativno iskanje višine tal je odvisno od lukenj v podatkih, zato je ob nekaterih predelih dokaj hitro, drugje, pri več luknjah, pa kar zahtevno. Pošiljanje je večinoma odvisno od knjižnice *civetweb* [16], saj se v tem koraku le preda že pripravljene podatke. V pošiljanje so vključene tudi škatle, ki se pošljejo iz predpomnilnika.

Gradnja k-d drevesa se izvaja s pomočjo optimizirane knjižnice *nanoflann* [41] in je odvisna od števila točk v oblaku. To potrdita trajanji gradnje drevesa vseh točk in manjšega števila točk tal, saj je prvo dvakrat počasnejše od drugega. Kvantizacija je odvisna od števila točk, a ni preveč zahtevna. Izračun stolpcev je odvisen samo od velikosti škatle, saj deluje samo na ravni blokov. Enako velja za zapolnitev zgradb, ki je sicer odvisna od števila blokov zgradb, vendar v praksi to ne pride do izraza.

Vzpostavitev *LASlib* [22] bralcev in branje ortofoto posnetkov se zgodita le ob novo obiskanem LIDAR kvadratu. Najbolj zahteven del pri teh je razširitev in dekodiranje PNG formata slike posnetkov. LIDAR kvadrati so načeloma dosti večji od škatel, zato se ta koraka opravita le redko in hitrost ni ključnega pomena. Izenačevanje vode deluje samo na principu površine, zato je lahko zelo hitro, saj preveri samo pripravljene stolpce. Poglabljanje vode je potrebno le pri škatlah z vodno površino, kjer se popravi le stolpce vodne površine, pri ostalih ni opravila.



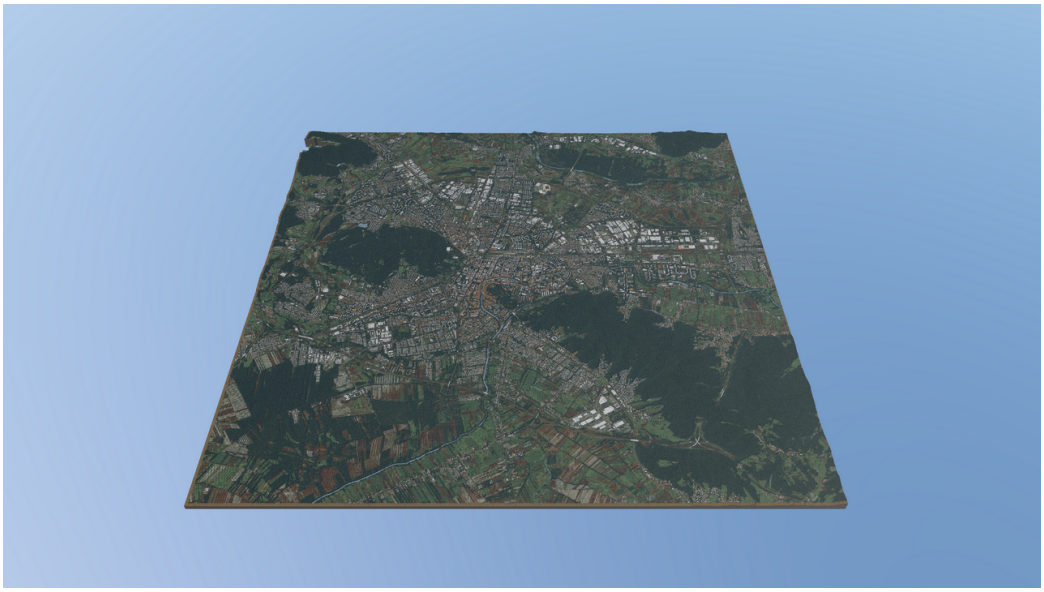
Slika 7.6: Pregled trajanja korakov strežnika skozi čas med upodabljanjem.



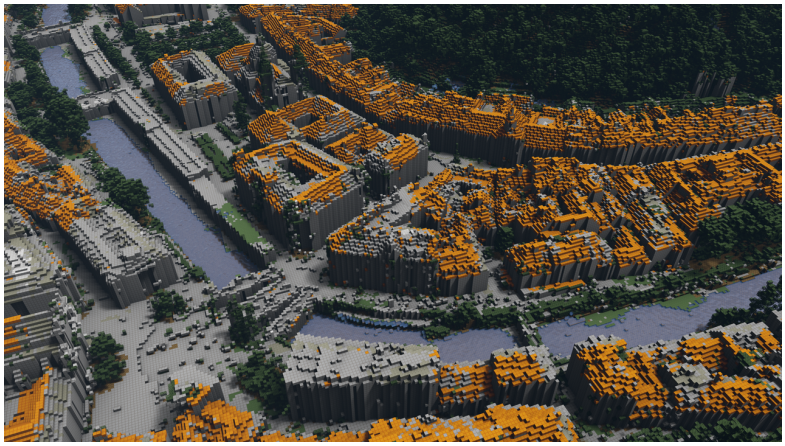
Slika 7.7: Podrobno trajanje korakov strežnika skozi čas med upodabljanjem.

7.3 Upodobitve

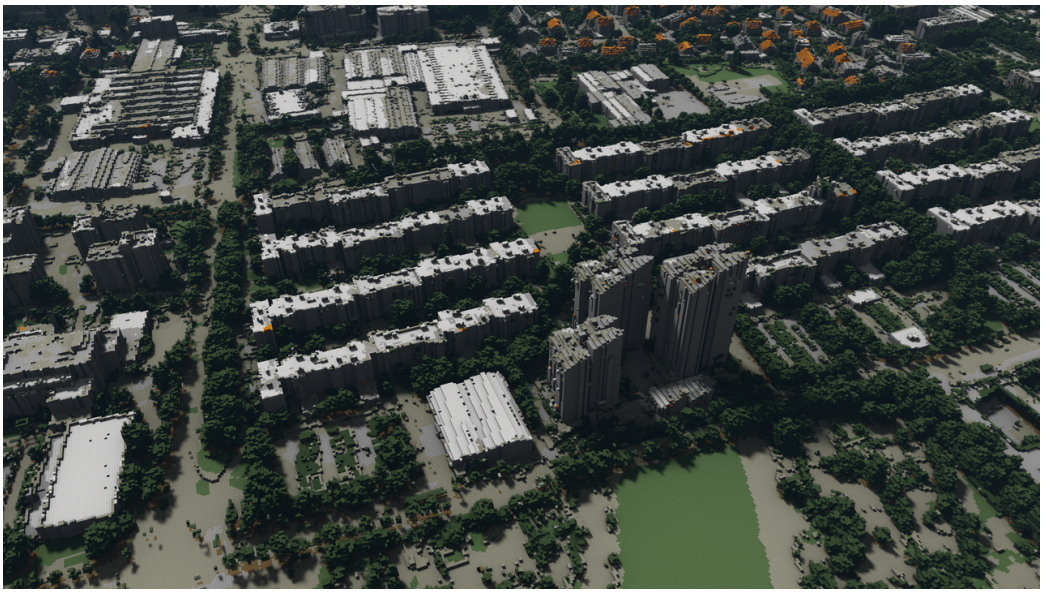
Sledijo izbrane slike, upodobljene s predstavljenimi orodji in programi.



Slika 7.8: Oddaljena upodobitev Ljubljane in okolice.



Slika 7.9: Prešernov trg in Tromostovje.

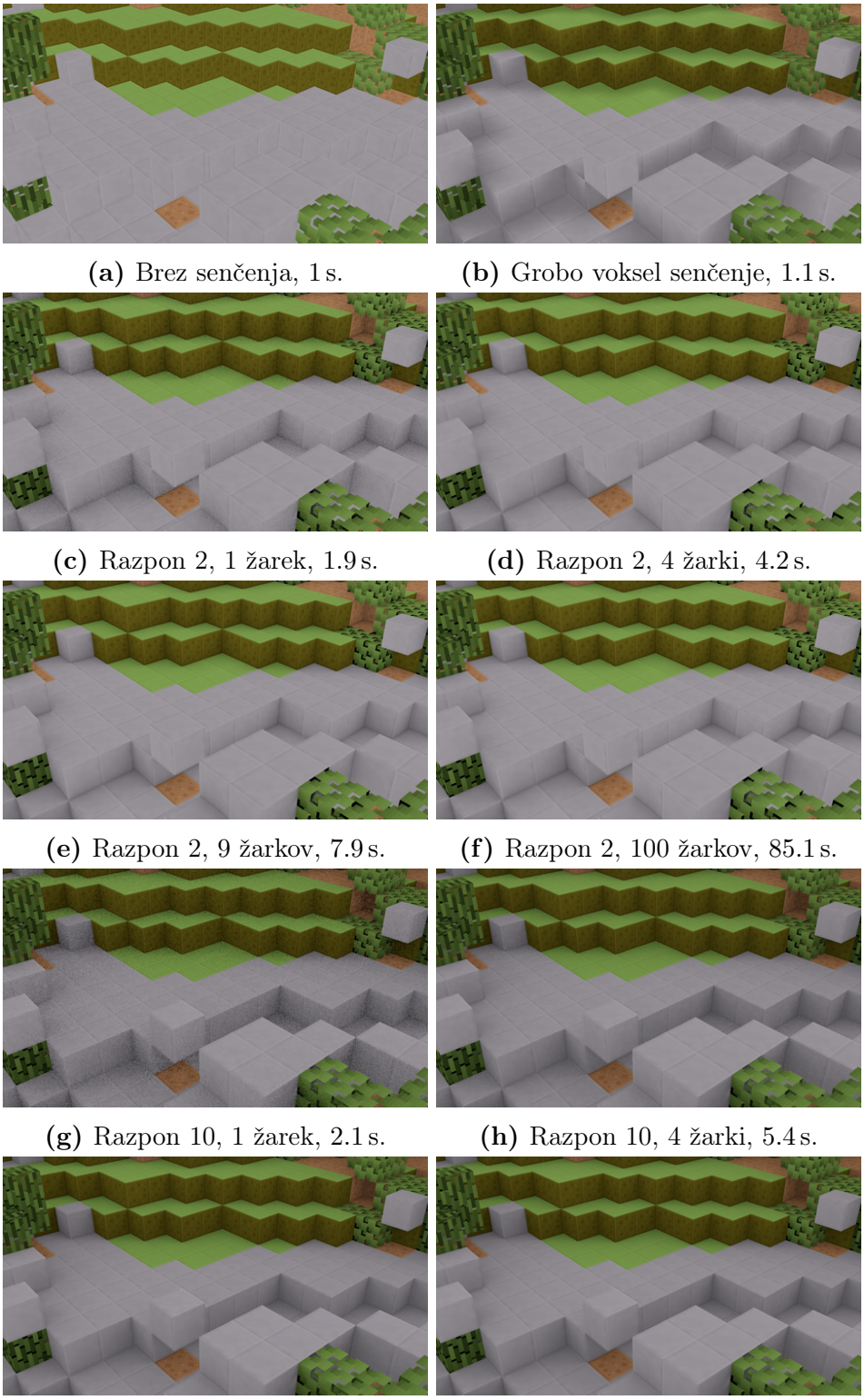


Slika 7.10: Stolpnice BS3.

Slika 7.11: Ljubljanski grad in okolica.

Slika 7.12: Jakopičevo sprehajališče.

Slika 7.13: Blejski otok.



(a) Brez senčenja, 1 s. **(b)** Grobo voksel senčenje, 1.1 s.

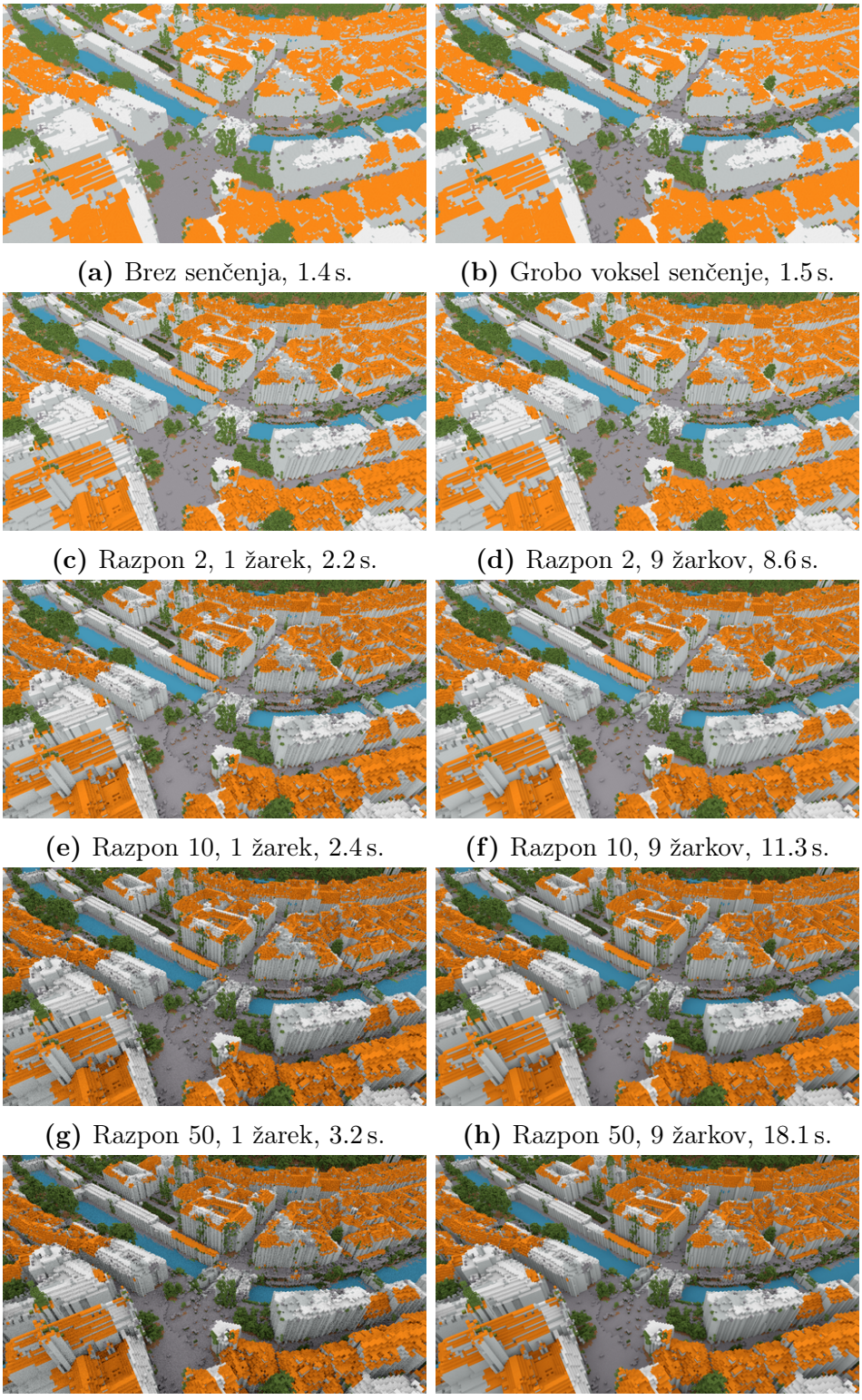
(c) Razpon 2, 1 žarek, 1.9 s. **(d)** Razpon 2, 4 žarki, 4.2 s.

(e) Razpon 2, 9 žarkov, 7.9 s. **(f)** Razpon 2, 100 žarkov, 85.1 s.

(g) Razpon 10, 1 žarek, 2.1 s. **(h)** Razpon 10, 4 žarki, 5.4 s.

(i) Razpon 10, 9 žarkov, 10.6 s. **(j)** Razpon 10, 100 žarkov, 111.6 s.

Slika 7.14: Primerjava različnih načinov in časa upodabljanja okoliškega senčenja bližnjih blokov.



(a) Brez senčenja, 1.4 s. **(b)** Grobo voksel senčenje, 1.5 s.

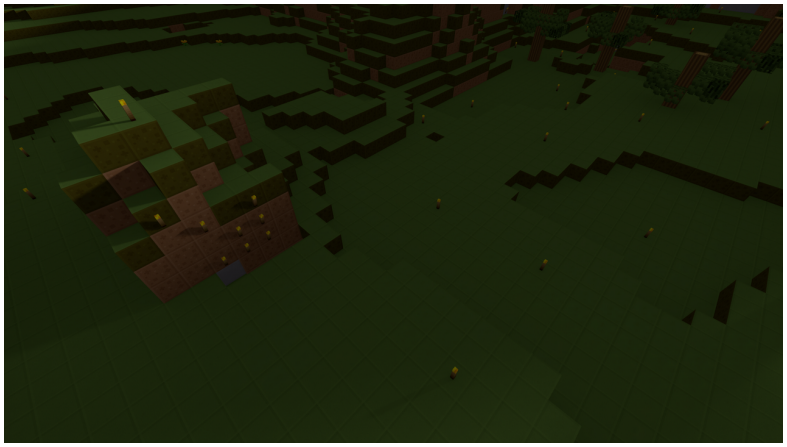
(c) Razpon 2, 1 žarek, 2.2 s. **(d)** Razpon 2, 9 žarkov, 8.6 s.

(e) Razpon 10, 1 žarek, 2.4 s. **(f)** Razpon 10, 9 žarkov, 11.3 s.

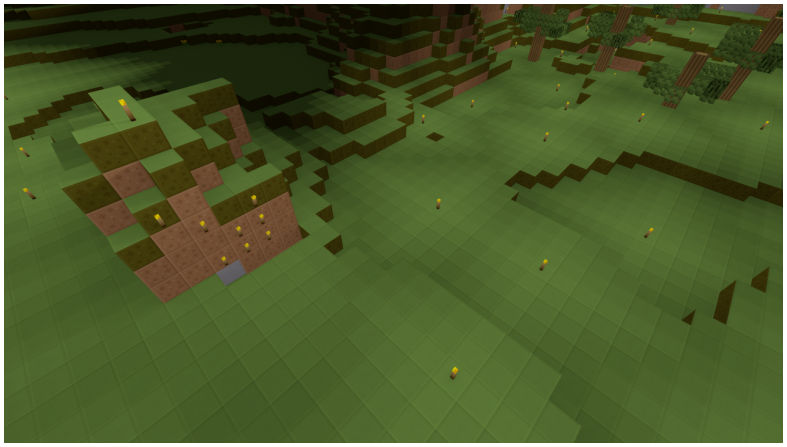
(g) Razpon 50, 1 žarek, 3.2 s. **(h)** Razpon 50, 9 žarkov, 18.1 s.

(i) Razpon 100, 1 žarek, 4.5 s. **(j)** Razpon 100, 9 žarkov, 20.3 s.

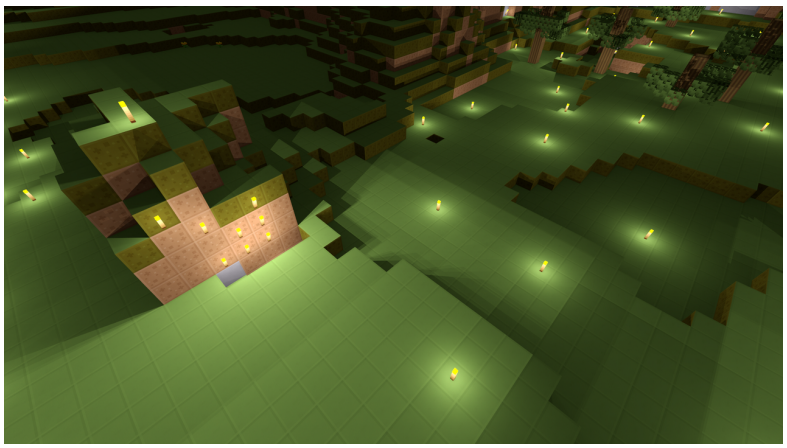
Slika 7.15: Primerjava različnih načinov in časa upodabljanja okoliškega senčenja oddaljenega pogleda.



(a) Brez razsvetljave blokov, 398 s.



(b) Groba voksel razsvetljava blokov, 406 s.



(c) Razsvetljava blokov z žarki, 445 s.

Slika 7.16: Primerjava načinov razsvetljave blokov s časi upodabljanja.

(a) Samo kvantizacija.

(b) Zapolnitev zgradb.

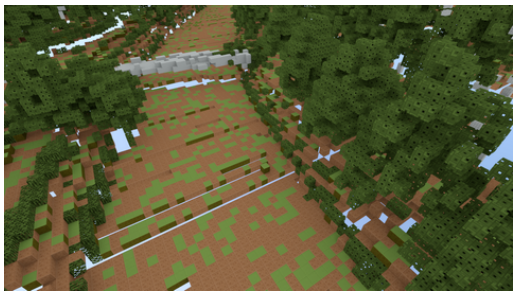
(c) Zapolnitev tal.

(d) Specializacija.

(e) Filtriranje.

(f) Izenačevanje in poglabljanje vodne površine.

Slika 7.17: Upodobljen oddaljen prizor Ljubljane po vsakem koraku strežnika.



(a) Samo kvantizacija.

(b) Zapolnitev zgradb.

(c) Zapolnitev tal.

(d) Specializacija.

(e) Filtriranje.

(f) Izenačevanje in poglabljanje vodne površine.

Slika 7.18: Upodobljen bližnji prizor Ljubljanice po vsakem koraku strežnika.

Poglavje 8

Sklep

V okviru diplomske naloge je bil razvit porazdeljen sledilnik žarkov, ki lahko upodablja virtualen svet igre Minecraft ali pa svet, ki je zgrajen iz podatkov laserskega skeniranja Slovenije. Razvita je bila skripta za prenos podatkov iz virov in njihovo pretvorbo. Implementiran je bil strežnik, ki iz pridobljenih podatkov gradi svet, ki ga nato upodobi sledilnik.

Sledilnik žarkov je primeren za manjše in srednje velike horizontalne razdalje, saj mora imeti vsak žarek, zaradi trenutne implementacije, naložene vse kose, skozi katere gre na svoji poti. S tem je pri večjih horizontalnih razdaljah sledilnik pomnilniško omejen. Ena izmed izboljšav bi bila možnost nadaljevanja sledenja žarka po prekinitvi njegove poti. Pri visokih pogledih ptičje perspektive te omejitve ni, saj vsak žarek obišče le manjše število kosov. Pri bolj zahtevnih prizorih je hitrost sledilnika večinoma odvisna od izbrane hierarhije predpomnilnikov.

Strežnik ima pri gradnji škatel tudi nekaj pomanjkljivosti. Klasifikacija in specializacija blokov je dokaj površna, možna izboljšava bi bila uporaba več vrst virov in boljši klasifikacijski model. Pri večjem pomanjkanju točk v oblaku (na primer sredi jezera) strežnik vrne prazno škatlo, lahko pa se bi zgradil približek tal iz ortofoto slik in višin tal *DMR*. Za nadaljnje delo bi se lahko strani zgradb, ki so trenutno puste, zapolnile iz georeferenciranih fotografskih posnetkov. Strežnik bi lahko ponujal tudi izbiro dela Slovenije

preko zemljevida, zgrajene škatle pa bi lahko izvozil kot tridimenzionalni model, ki bi ga bilo na primer možno natisniti s 3D tiskalnikom.

Izboljšani nov program sledilnika bi se lahko v interes hitrosti izvajal heterogeno na grafični kartici in centralni procesni enoti. S pomočjo urejanja žarkov po položaju in smeri bi se lahko različni predpomnilniki bolje izrabili. Polje oddaljenosti bi pohitrilo skoke skozi prazne prostore.

Področji sledenja žarkov oziroma upodabljanja in obdelave lasersko zajetih podatkov sta zelo razširjeni in tako ponujata neštete možne nove metode, izboljšave, nadgradnje in dodatke.

Literatura

[1] *Adobe® Flex® Software Development Kit (SDK)*. Angl. Adobe Systems Incorporated. 2016. URL: http://www.adobe.com/go/flex_sdk/ (pridobljeno 2. 7. 2016).

[2] J. Amanatides, A. Woo in sod. "A fast voxel traversal algorithm for ray tracing". Angl. V: *Eurographics*. Zv. 87. 3. 1987, str. 3–10. URL: http://www.cse.chalmers.se/edu/year/2015/course/TDA361/grid.pdf (pridobljeno 2. 7. 2016).

[3] S. E. Anderson. *Bit Twiddling Hacks*. Angl. 2009. URL: https://graphics.stanford.edu/~seander/bithacks.html#InterleaveBMN (pridobljeno 2. 7. 2016).

[4] S. Barrett in sod. *stb. Single-file public domain libraries for C/C++*. Angl. 2016. URL: https://github.com/nothings/stb (pridobljeno 2. 7. 2016).

[5] *Blender. Home of the Blender project - Free and Open 3D Creation Software*. Angl. Blender Foundation. 2016. URL: https://www.blender.org/ (pridobljeno 1. 7. 2016).

[6] M. Bostock, J. Davies in sod. *D3: Data-Driven Documents*. Angl. 2016. URL: https://d3js.org/ (pridobljeno 2. 7. 2016).

[7] *CMake. CMake is an open-source, cross-platform family of tools designed to build, test and package software.* Angl. Kitware. 2016. URL: https://cmake.org/ (pridobljeno 2. 7. 2016).

[8] Codermind team. *Ray Tracing Tutorial*. Angl. URL: https://www.ics.uci.edu/~gopi/CS211B/RayTracing%20tutorial.pdf (pridobljeno 2. 7. 2016).

[9] B. De Greve. *Reflections and refractions in ray tracing*. Angl. Stanford University, 2006. URL: http://graphics.stanford.edu/courses/cs148-10-summer/docs/2006--degreve--reflection_refraction.pdf (pridobljeno 2. 7. 2016).

[10] B. Egner, Moonlight63, J. Horsley in sod. *Jmc2Obj. A Minecraft to .Obj exporter written in Java*. Angl. 2016. URL: http://www.jmc2obj.net/ (pridobljeno 1. 7. 2016).

[11] ephtracy. *MagicaVoxel. A free lightweight 8-bit voxel editor and interactive path tracing renderer*. Angl. 2016. URL: https://ephtracy.github.io/ (pridobljeno 1. 7. 2016).

[12] FalconNL. *mc2obj. Minecraft to OBJ (and PRT) converter*. Angl. 2011. URL: https://github.com/FalconNL/mc2obj (pridobljeno 1. 7. 2016).

[13] *FAQ - HDR images for Photography*. Angl. HDRsoft. URL: http://www.hdrsoft.com/resources/dri.html (pridobljeno 2. 7. 2016).

[14] *Flash Player and Adobe AIR feature list*. Angl. Adobe Systems Incorporated. 2016. URL: http://www.adobe.com/devnet/articles/flashplayer-air-feature-list.html (pridobljeno 2. 7. 2016).

[15] *FlashDevelop*. Angl. 2016. URL: http://flashdevelop.org/ (pridobljeno 2. 7. 2016).

[16] F-Secure Corporation, T. Davis, S. Lyubka in The CivetWeb developers. *CivetWeb. Embedded C/C++ web server*. Angl. 2016. URL: https://github.com/civetweb/civetweb (pridobljeno 2. 7. 2016).

[17] S. Gustavson. "Simplex noise demystified". Angl. V: *Linköping University, Linköping, Sweden, Research Report* (2005). URL: http://webstaff.itn.liu.se/~stegu/simplexnoise/simplexnoise.pdf (pridobljeno 2. 7. 2016).

[18] J. Halliday. *mkdirp. Recursively mkdir, like 'mkdir -p', but in node.js*. Angl. 2015. URL: https://github.com/substack/node-mkdirp (pridobljeno 2. 7. 2016).

[19] D. Heeger. "Perception Lecture Notes: Light/Dark Adaptation". Angl. New York, 2006. URL: http://www.cns.nyu.edu/~david/courses/perception/lecturenotes/light-adapt/light-adapt.html (pridobljeno 2. 7. 2016).

[20] *Horizontalni državni koordinatni sistem (D96/TM). Razlika med starim in novim KS*. MOP - Geodetska uprava Republike Slovenije. 2016. URL: http://www.e-prostor.gov.si/si/zbirke_prostorskih_podatkov/drzavni_koordinatni_sistem/horizontalni_drzavni_koordinatni_sistem_d96tm/ (pridobljeno 2. 7. 2016).

[21] *How does LiDAR work? The science behind the technology*. Angl. Bluesky International Limited. 2016. URL: http://www.lidar-uk.com/how-lidar-works/ (pridobljeno 2. 7. 2016).

[22] M. Isenburg. *LASlib. LASlib (with LASzip) is a C++ programming API for reading / writing LIDAR data stored in standard LAS or in compressed LAZ format (1.0 - 1.3)*. Angl. rapidlasso GmbH. 2016. URL: https://github.com/LAStools/LAStools/tree/master/LASlib (pridobljeno 2. 7. 2016).

[23] M. Isenburg. *lasliberate. A simple tool to free LiDAR locked-up in ESRI's "Optimized LAS"*. Angl. rapidlasso GmbH. 2015. URL: https://github.com/LASliberator/lasliberate (pridobljeno 2. 7. 2016).

[24] M. Isenburg. *LASzip. The LASzip LiDAR compressor packaged as a simple stand-alone DLL for easy inclusion of compression and decompression functionality into other software*. Angl. rapidlasso GmbH. 2016. URL: https://github.com/LAStools/LAStools/tree/master/LASzip (pridobljeno 2. 7. 2016).

[25] *Izvedba laserskega skeniranja Slovenije. Blok 35 - tehnično poročilo o izdelavi izdelkov*. Geodetski inštitut Slovenije. 21. avg. 2015. URL: http://gis.arso.gov.si/related/lidar_porocila/b_35_izdelava_izdelkov.pdf (pridobljeno 2. 7. 2016).

[26] M. Knight. *node-dbf. An efficient dBase DBF file parser written in pure JavaScript.* Angl. 2016. URL: https://github.com/abstractvector/node-dbf (pridobljeno 2. 7. 2016).

[27] A. W. Kristensen. *µjson. µjson is a a small, C++11, UTF-8, JSON library.* Angl. 2015. URL: https://bitbucket.org/awangk/ujson (pridobljeno 2. 7. 2016).

[28] E. P. Lafortune in Y. D. Willems. "Bi-directional Path Tracing". Angl. V: Compugraphics. (1993). Ur. H. P. Santo. Katholieke Universiteit Leuven. 1993, str. 145–153. URL: http://graphics.cs.kuleuven.be/publications/BDPT/ (pridobljeno 2. 7. 2016).

[29] *LAS Specification Version 1.4 – R6.* Angl. The American Society for Photogrammetry & Remote Sensing. 15. avg. 2011. URL: http://www.asprs.org/wp-content/uploads/2010/12/LAS_1-4_R6.pdf (pridobljeno 2. 7. 2016).

[30] *LAStools.* Angl. rapidlasso GmbH. 2015. URL: https://rapidlasso.com/lastools/ (pridobljeno 1. 7. 2016).

[31] *LIDAR.* Ministrstvo za okolje in prostor RS. 2015. URL: http://evode.arso.gov.si/indexd022.html?q=node/12 (pridobljeno 2. 7. 2016).

[32] *LiDAR LiVE.* GeMMA Lab, Fakulteta za elektrotehniko, računalništvo in informatiko (FERI). 2009. URL: http://gemma.uni-mb.si/lidarlive/ (pridobljeno 1. 7. 2016).

[33] J. Lukic in sod. *Semantic UI. Semantic empowers designers and developers by creating a shared vocabulary for UI.* Angl. Semantic Organization. 2016. URL: http://semantic-ui.com/ (pridobljeno 2. 7. 2016).

[34] G. van Maren. *Esri Introduces Optimized LAS.* Angl. Esri. 2014. URL: https://blogs.esri.com/esri/arcgis/2014/01/10/esri-introduces-optimized-las/ (pridobljeno 2. 7. 2016).

[35] C. McMahon. *async. Async utilities for node and the browser.* Angl. 2016. URL: https://github.com/caolan/async (pridobljeno 2. 7. 2016).

[36] *Minecraft. Minecraft is a game about placing blocks and going on adventures.* Angl. Mojang Synergies AB. 2016. URL: `https://minecraft.net` (pridobljeno 3. 7. 2016).

[37] *Ministrstvo je zagotovilo brezplačen in javen dostop vsem do podatkov LIDAR snemanj za območje celotne Slovenije.* Ministrstvo za okolje in prostor RS. 15. sep. 2015. URL: `http://www.mop.gov.si/si/medijsko_sredisce/novica/article/1328/6239/2bfa6fcd2f1e933d92beb7c068f11695/` (pridobljeno 2. 7. 2016).

[38] D. Mongus, M. T. Čekada in B. Žalik. "Analiza samodejne metode za generiranje digitalnih modelov reliefa iz podatkov lidar na območju Slovenije". V: *Geodetski vestnik* 57.2 (2013), str. 245.

[39] B. Monteverde, R. Hu in sod. *NVD3. A reusable D3 charting library.* Angl. Novus Partners, Inc. 2016. URL: `http://nvd3.org/` (pridobljeno 2. 7. 2016).

[40] *Mreža listov za celo Slovenijo (D96TM).* 2015. URL: `http://gis.arso.gov.si/related/lidar_porocila/lidar_fishnet_D96TM.zip` (pridobljeno 2. 7. 2016).

[41] M. Muja, D. G. Lowe in J. L. Blanco. *nanoflann. A C++ header-only library for Nearest Neighbor (NN) search wih KD-trees.* Angl. 2016. URL: `https://github.com/jlblancoc/nanoflann` (pridobljeno 2. 7. 2016).

[42] *Navodila za prenos podatkov LIDAR.* Ministrstvo za okolje in prostor RS. 2015. URL: `http://evode.arso.gov.si/indexd697.html?q=node/32` (pridobljeno 2. 7. 2016).

[43] *Node.js. Node.js ® is a JavaScript runtime built on Chrome's V8 JavaScript engine.* Angl. Node.js Foundation. 2016. URL: `https://nodejs.org/` (pridobljeno 2. 7. 2016).

[44] S. O'Neil. "Accurate atmospheric scattering". Angl. V: *GPU Gems* 2 (2005), str. 253–268.

[45] *OpenEXR*. Angl. Industrial Light & Magic. URL: http://www.openexr.com (pridobljeno 2. 7. 2016).

[46] J. Öqvist. *Chunky. Chunky is a Minecraft mapping and rendering tool*. Angl. 2016. URL: http://chunky.llbit.se/ (pridobljeno 1. 7. 2016).

[47] R. Ostafichuk. *DBFEngine. Simple Engine to read and write FoxPro .dbf files*. Angl. 2016. URL: https://github.com/bramhe/DBFEngine (pridobljeno 2. 7. 2016).

[48] S. Pečnik, B. Žalik in D. Mongus. "Interaktivno orodje za obdelavo podatkov LiDAR". Diplomska naloga univerzitetnega študijskega programa. Fakulteta za elektrotehniko, računalništvo in informatiko (FERI), Univerza v Mariboru, 2009. URL: https://dk.um.si/IzpisGradiva.php?id=12795 (pridobljeno 1. 7. 2016).

[49] J. Resig, D. Methvin, T. Willison in sod. *jQuery. The Write Less, Do More, JavaScript Library*. Angl. The jQuery Foundation. 2016. URL: https://jquery.com/ (pridobljeno 2. 7. 2016).

[50] H. Seetzen in sod. "High dynamic range display systems". Angl. V: *ACM Transactions on Graphics (TOG)* 23.3 (2004), str. 760–768. URL: http://anyhere.com/gward/papers/Siggraph04.pdf (pridobljeno 2. 7. 2016).

[51] *Simulating the Colors of the Sky. Atmospheric Scattering*. Angl. Scratchapixel. 2012. URL: http://www.scratchapixel.com/old/lessons/3d-advanced-lessons/simulating-the-colors-of-the-sky/atmospheric-scattering/ (pridobljeno 2. 7. 2016).

[52] *Spletni portal eVode*. Ministrstvo za okolje in prostor RS. 2015. URL: http://evode.arso.gov.si (pridobljeno 2. 7. 2016).

[53] M. Stokes, M. Anderson, S. Chandrasekar in R. Motta. *A Standard Default Color Space for the Internet — sRGB*. Angl. World Wide Web Consortium. 1996. URL: http://www.w3.org/Graphics/Color/sRGB (pridobljeno 2. 7. 2016).

[54] *Understanding Gamma Correction*. Angl. Cambridge in Colour. 2016. URL: http://www.cambridgeincolour.com/tutorials/gamma-correction.htm (pridobljeno 2. 7. 2016).

[55] Ventero in sod. *amf-cpp. A C++ implementation of the Action Message Format (AMF3)*. Angl. 2016. URL: https://github.com/Ventero/amf-cpp (pridobljeno 2. 7. 2016).

[56] *Visual Studio Code. Code editing. Redefined*. Angl. Microsoft. 2016. URL: https://code.visualstudio.com/ (pridobljeno 2. 7. 2016).

[57] *Visual Studio Community*. Angl. Microsoft. 2015. URL: https://www.visualstudio.com/en-us/products/visual-studio-community-vs.aspx (pridobljeno 2. 7. 2016).

[58] A. Völk. *piehole. A Minecraft 1.0 texture pack*. Angl. Impressum. 2011. URL: http://www.piehole.de/ (pridobljeno 4. 7. 2016).

[59] J. Walnes in D. Noakes. *Smoothie Charts. A JavaScript Charting Library for Streaming Data*. Angl. 2014. URL: http://smoothiecharts.org/ (pridobljeno 2. 7. 2016).

[60] J. Wright. *mcobj. Minecraft to OBJ (and PRT) converter*. Angl. 2011. URL: https://github.com/quag/mcobj (pridobljeno 1. 7. 2016).